NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

**THREE DIMENSIONAL GUIDANCE
FOR THE NPS
AUTONOMOUS UNDERWATER VEHICLE**

by

Christopher Magrino

September 1991

Thesis Advisor: Yutaka Kanayama

REPORT DOCUMENTATION PAGE

a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited
b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: omputer Science Dept. / aval Postgraduate School; 6b. OFFICE SYMBOL (if applicable): CS	7a. NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
c. ADDRESS (City, State, and ZIP Code): lonterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
HREE DIMENSIONAL GUIDANCE FOR THE NPS AUTONOMOUS UNDERWATER VEHICLE (U)

12. PERSONAL AUTHOR(S)
lagrino, Christopher (NMN)

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
laster's Thesis	From: 6/90 To: 9/91	1991 September 26	116

16. SUPPLEMENTARY NOTATION: The views expressed in this thesis are those of the author and do not reflect the official olicy or position of the Department of Defense or the United States Government.

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
spatial tracking, guidance, cross track navigation, autonomous undewater vehicle, AUV, control system

19. ABSTRACT (Continue on reverse if necessary and identify by block number)
The Naval Postgraduate School is currently conducting research in the area of autonomous underwater vehicles. n support of this research, the school has developed a testbed vehicle and graphic simulation. One of the major thrusts f the project is the development of a control system.

This work explores the implementation and testing of a guidance scheme proposed by Kanayama called spatial acking. The method is evaluated with and without consideration for AUV dynamics. Spatial tracking is also com- ared with an earlier guidance scheme attributed to Kanayama known as cross track guidance.

The NPS AUV testbed vehicle and simulator are also described within this work.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL: utaka Kanayama	22b. TELEPHONE (Include Area Code): (408) 646-2095 / 22c. OFFICE SYMBOL: CSKa

FORM 1473, 84 MAR — 83 APR edition may be used until exhausted; All other editions are obsolete

Approved for public release; distribution is unlimited

THREE DIMENSIONAL GUIDANCE FOR THE NPS AUTONOMOUS UNDERWATER VEHICLE

by
Christopher Magrino
Lieutenant, United States Navy
B.A., Miami University, 1985

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September, 1991

ABSTRACT

The Naval Postgraduate School is currently conducting research in the area of autonomous underwater vehicles. In support of this research, the school has developed a testbed vehicle and graphic simulation. One of the major thrusts of the project is the development of a control system.

This work explores the implementation and testing of a guidance scheme proposed by Kanayama called spatial tracking. The method is evaluated with and without consideration for AUV dynamics. Spatial tracking is also compared with an earlier guidance scheme attributed to Kanayama known as cross track guidance.

The NPS AUV testbed vehicle and simulator are also described within this work.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A. BACKGROUND

In the last several years, research in the area of autonomous submersibles has become an important issue within the Department of Defense. It is projected that autonomous underwater vehicles will be capable of conducting a variety of missions such as underwater surveillance, tracking, mapping, mine laying, and even offensive anti-submarine operations. There are some obvious advantages associated with using an unmanned submersible in missions of this nature. First, an AUV (autonomous underwater vehicle) may perform tasks which are considered too dangerous for humans to accomplish. Second, they may operate without regard to the physiological limitations that constrain manned vehicles.

However, there are also problems associated with the use of unmanned, unthethered vehicles. First, once the vehicle is launched on a mission, the vehicle is on its own. The operator must rely upon the vehicle's control software to carry out the assigned mission. This, in itself, is a complex problem that requires detailed programs to cover every possible situation. To add to this complexity, the vehicle must also possess the ability to conduct its mission even in the presence of unforeseen difficulties.

One particularly important aspect of control software is the guidance system. The guidance system is responsible for taking information on current vehicle position and orientation and comparing it with desired vehicle position and orientation. The resulting information is used to either maintain the current path (if current and desired locations are the same) or correct to achieve the desired path (if current and desired locations are different).

The Naval Postgraduate School has been conducting research in an attempt to solve the problems mentioned above. The school has a working testbed vehicle that is supported by a graphic simulation. Both the vehicle and the simulation will be described within this work.

In support of the NPS AUV project, the major thrust of this work is to implement and test a three dimensional guidance scheme developed by Kanayama. We call this new guidance scheme spatial tracking. Within the scope of this work, the spatial tracking guidance system is installed and tested on the simulator version of the AUV. Due to the modularity of design, the software may be easily installed in the testbed vehicle later on in the developmental stage.

Another important topic discussed within this work is the examination of an earlier guidance scheme by Kanayama known as cross track guidance. This scheme will then be compared with spatial tracking.

B. THESIS ORGANIZATION

The remainder of this work is organized as follows.

Chapter II is a review of previously published work relating to guidance systems for autonomous vehicles. These works have served as building blocks for this research and their effects on it are summarized.

Chapter III is a description of the AUV simulator using the spatial tracking method of guidance. The data structures, modules, and data stores are presented.

Chapter IV is a description of the AUV system. First the actual testbed vehicle is described. The chapter concludes with a section that compares the testbed vehicle with the simulator discussed in Chapter III. Major differences between the two are highlighted.

Chapter V presents cross track guidance and spatial tracking. The chapter begins with the presentation of cross track guidance and continues with the formal description of spatial tracking. These results are independent of any vehicle constraints.

Chapter VI presents the test results of the spatial tracking guidance scheme as they apply to the NPS AUV Simulator. The chapter begins with the presentation of two topics that are important when the guidance system is applied to the simulator. The first explores the use of limiters to prevent excessive curvature values. The second topic is a discussion of changing reference paths. The chapter concludes with the presentation of the results of test cases that are run on the simulator.

Chapter VII is the conclusion of this work. The scientific value of this work is evaluated and its short comings are identified. Also, the areas that need to be more thoroughly explored are identified in the hopes that future research will be conducted in these areas.

II. REVIEW OF PREVIOUS WORK

This chapter provides an overview of previously published work relating to guidance systems for autonomous vehicles. These works have served as building blocks for this research and their effects on it are summarized below.

A. YAMABICO-11

In [KANAYAMA 88], [KANAYAMA 90], and [HARTMAN 89], a guidance system was proposed for the Yamabico-11 autonomous mobile land vehicle. In [KANAYAMA 91], a set of locomotion functions were developed for implementation on this vehicle. The guidance system of the vehicle used reference and current postures for smooth and precise positional control which made dynamic posture correction feasible. The product of the guidance system was rate information which was then sent on to the autopilot where this rate information was turned into corrections to the positioning actuators of the vehicle. The vehicle's stability was proved using Liapunov functions. The guidance system used cubic spirals as the desired path to follow between postures which had the effect of minimizing jerk between waypoints. The minimization of jerk is a very desirable feature for a guidance system of an underwater vehicle. Since this vehicle was for terrestrial uses, the guidance system was designed for two dimensional applications only. However, due to the simplicity of this guidance system, an extension to three dimensional applications appears to be practicable. The concept of a posture that was introduced in this work has been expanded to a three dimensional posture for the NPS AUV.

B. WOODS HOLE RPV

The Woods Hole Oceanographic Institute has been conducting research in the field of underwater vehicles for many years. In the past they have developed such vehicles as

JASON and JASON Jr. that have been successfully used to conduct sonar, video, and electronic photographic surveys of underwater objects in the open ocean. These two remotely operated vehicles require precise control which is sometimes difficult to provide due to their nonhydrodynamic shapes. To ensure continuing evolution of advanced control systems for future development at Woods Hole, the RPV test bed vehicle has also been developed. This vehicle's components are modeled after the JASON series ROVs so as to serve as a research model for evolution of the operational ROVs.

Unlike the NPS AUV, the Woods Hole RPV uses five thrusters (four lateral and one vertical) as its principle means of movement. This scheme of propulsion introduces several control problems. In [YOERGER 91], a trajectory control scheme using adaptive sliding control techniques was proposed to alleviate these problems. The adaptive control allows the operational system to be robust to unanticipated changes in the vehicle's dynamic parameters. The uncertainty-performance trade-off that is explicitly available with sliding control techniques allows the formulation of adaptation laws for a wide class of nonlinear systems based upon Liapunov stability considerations.

C. UCSB SIMULATION

The University of California at Santa Barbara has conducted research in three dimensional guidance for underwater autonomous vehicles. In [SAVANT 90], a Liapunov function approach has been used to develop a non-linear feedback control scheme for non-holonomic vehicles. The work is an expansion of the two dimensional algorithms introduced by Kanayama for the Yamabico-11 mobile land vehicle [KANAYAMA 90]. This research applies three dimensional guidance equations to a computer simulated submersible represented by a point mass. In addition, the effects of side slip were not considered as all vehicle motion was assumed along the longitudinal axis of the vehicle. As

in the Yamabico-11, the guidance system output rate information which was used by the autopilot to attempt to regain the reference posture.

D. NPS AUVII

The current version of the NPS AUV project is in version III. Its predecessor was the AUV II. In [CLOUTIER 90], the guidance system used cubic spirals as the desired path to follow between waypoints. Waypoints were positionally restricted to a three dimensional grid where allowable waypoints were positioned at the intersections of the x, y, and z planes. Actual grid size was to be determined by actual vehicle maneuverability from in-water tests. As in the Yamabico-11, the guidance system used reference and current postures to determine error postures. The error posture information was used to determine a cubic spiral path between waypoints that was smooth and that minimized jerk. This cycle was performed on a 10 hertz frequency and generated an intermediate waypoint on each iteration. Each intermediate waypoint was to fit the cubic spiral path between waypoints and was sent to the autopilot for processing.

The evolution of the AUV III will use portions of the guidance scheme presented above. The concept of a current posture is still valid. This current posture will be used to generate a spatial posture. With AUV III, we shall explore the use of spatial tracking as a means of guidance. Chapter V will discuss this method in detail.

III. NPS AUV III SIMULATOR

The Naval Postgraduate School has been developing a small autonomous underwater vehicle to be used for research purposes by faculty and students of the school. The major thrusts of the project are in the areas of mission planning, mission analysis, mission execution, and post mission data analysis [HEALY 90]. The actual vehicle is supported by a simulation installed on Silicon Graphics Iris 4D workstations [JUREWICZ 90]. The simulator reproduces near-actual vehicle responses and is under continued development so as to perform exactly like the testbed vehicle. The user's manual for this simulator is included in this work as Appendix C.

This chapter will discuss the simulator in depth. It should be pointed out that the required software modules for the two types of guidance systems discussed within this work are considerable different. Since the guidance system used by the simulator is spatial tracking, the simulator architecture presented here applies only to the spatial tracking guidance scheme. See Figure 3-1 for the data flow diagram that describes the simulator architecture.

A. DATA STRUCTURES

1. Postures

A posture is a data structure that holds position and orientation information that describes the six degrees of freedom for the AUV. Two types of postures are discussed in regards to the AUV Simulator:

* Current Posture

* Spatial Posture

Figure 3-1 **AUV Simulator Data Flow Diagram**

The current posture describes the actual position and orientation of the vehicle. It does this by holding a value for each element of the six degrees of freedom as represented as follows:

$$P_C = (x, y, z, \phi, \theta, \psi) \qquad \text{(Eq. 3-1)}$$

where, x, y, and z are the positional displacements of a vehicle from a predefined origin in the longitudinal, rotational, and vertical planes respectively, and ϕ, θ, and ψ are the angular displacements from the longitudinal axis in pitch, roll, and yaw respectively.

The makeup of a spatial posture is slightly different from the makeup of a current posture. A spatial posture holds the position and orientation information, but also holds additional attributes known as curvature. Curvature is a product of the guidance system and will be explained in detail in Chapter V. The six degrees of freedom and the curvature are expressed as follows:

$$P_S = (x, y, z, \phi, \theta, \psi, \kappa, \Gamma) \qquad \text{(Eq. 3-2)}$$

where the first six elements are as above, and κ is the curvature of the projection of a path in the x, y plane and Γ is the curvature of the projection of a path in the x, z plane.

Additionally, during the discussion of cross track guidance in Chapter V, reference and error postures are introduced. These two postures will be defined in the discussion of cross track guidance.

2. Waypoints

A waypoint is a three dimensional position expressed as:

$$(x, y, z) \qquad \text{(Eq. 3-3)}$$

In addition, depending upon the type of path that is to be represented by a waypoint, the data structure may contain orientation, curvature, and/or radius information. This additional information must be derived from the values of equation 3-3. Waypoints come from either the user specified mission requirements or from the obstacle avoider.

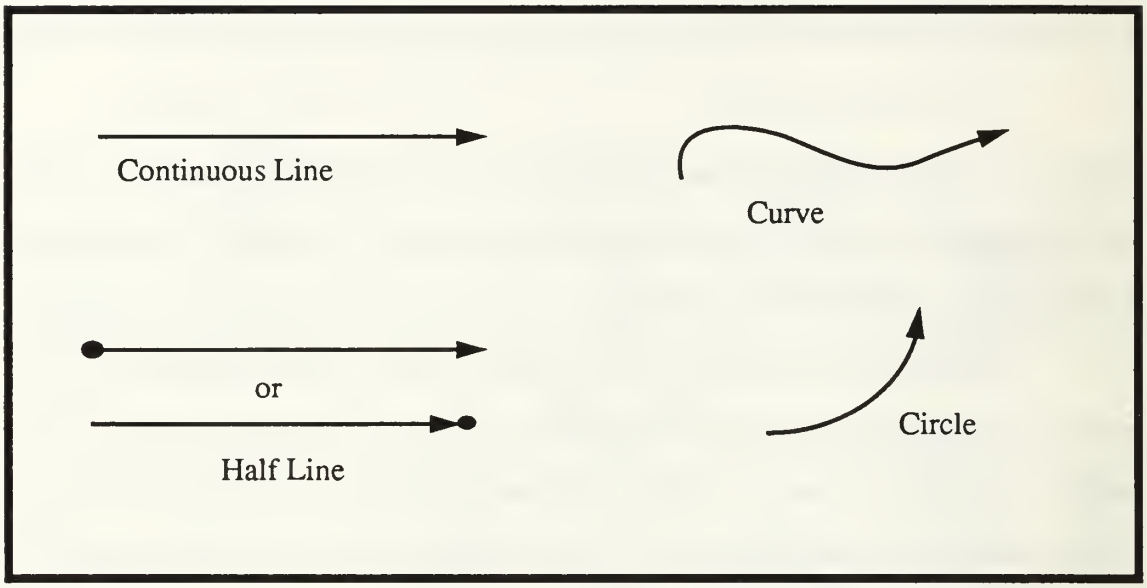

Figure 3-2 **Reference Path Representation**

Waypoints from the obstacle avoider override a waypoint specified by the user. In either case, they are used the same way by the guidance system. Waypoint are used to form a reference path. By linking adjacent waypoints with rays, a continuous path is formed. These rays are directed and may take on one of four different forms:

* Continuous line
* Half line
* Circle
* Curve

See Figure 3-2 for a graphical description of these four rays.

A continuous line is represented by a planar position and orientation in the x, y and x, z planes. This line has no beginning or end points. A half line is represented in the same manner as a continuous line; however, the point that specifies the line also specifies an end point. A circle may be described in two ways. The first way is to use a planar position that specifies the center of the circle and include a radius value. The sign of the

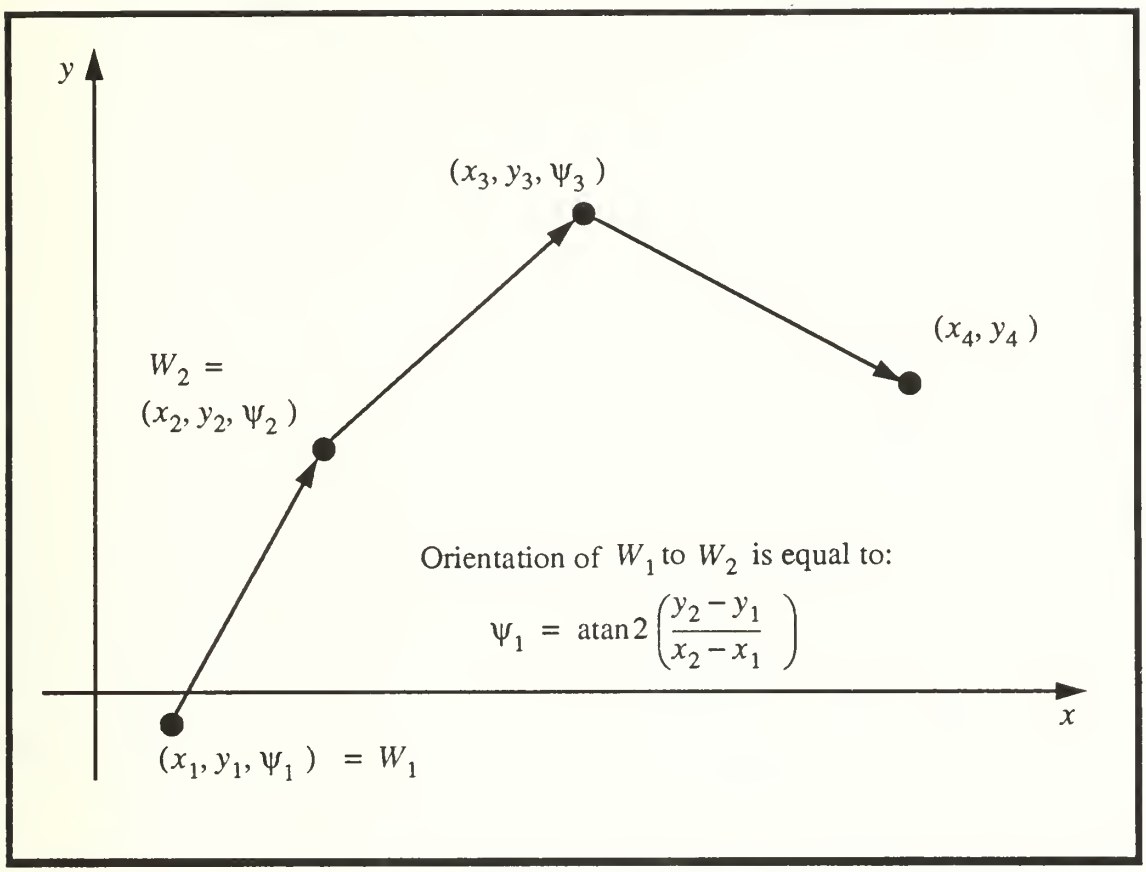

Figure 3-3 **Reference Path Representation Using Half Lines**

radius determines the direction. A positive sign signifies a counterclockwise movement while a negative sign indicates a clockwise movement. A circle may also be represented by a planar position located on the circumference with curvature values in the x, y and x, z planes. Again, the sign of the curvature values indicate direction. The use of these two different representations for a circle are dictated by the circumstances. For example, the former representation would probably be more useful when avoiding obstacles, while the latter may be a better representation when switching paths. A curve may be represented using planar position and orientation. A curvature value with respect to distance from the planar position is also needed to specify curvature at a particular point along the curve.

The orientation between waypoints is specified in the x, y plane by taking the arc tangent of the difference between the x and y coordinates. To add a third dimension, the orientation of this ray is also specified in the x, z plane by taking the arc tangent of the difference between the x and z coordinates. Figure 3-3 gives a graphical description of this process in the x, y plane using directed half lines.

B. MODULES

The guidance module and the other three modules described in this section are the heart of the AUV simulator's control system. These explanations are provided so as to give one a better understanding of the interaction between the guidance system and the other three modules. Although the descriptions provided here apply only to the simulator version of the AUV, only minor modification would be necessary to install one or more of these modules in the testbed vehicle used at NPS. See the AUV Simulator Data Flow Diagram (Figure 3-1) for a complete breakdown of the simulator architecture.

1. Guidance

The guidance system is a key component of an autonomous vehicle. This system is responsible for directing the vehicle from waypoint to waypoint and also, when the vehicle's position is determined to be in error, its purpose is to calculate corrections to return the vehicle to the reference path.

There are several different types of guidance schemes that may be employed. Two of these schemes are discussed in Chapter V. For the purposes of this work, we will be implementing a guidance scheme developed by Kanayama that we call spatial tracking. This method compares the current spatial position of the vehicle with the reference path and determines a smooth path to travel. See Chapter V for a complete description of this

method. See Appendix B for a listing of the C program that implements this method on the AUV simulator.

2. Navigator

The navigator's main function is to read the sensors of the AUV and determine the exact position of the vehicle. On the NPS AUV, a sensor suite consisting of a directional gyro, a vertical gyro system, a three axis rate gyro system, and a three axis translational accelerometer are simulated to aid in fixing the vehicle's position. In [HEALY 90], the testbed vehicle was described to have this particular configuration. In [FLOYD 91a], pattern matching is proposed as a method of positional updating. In this method, range and bearing information received from the sensors is loaded into a least-squares-fit algorithm to determine surfaces of polyhedrons. These computed surfaces are then compared with the known environment model to try and determine the vehicle's actual location.

Once the actual vehicle location has been determined, the navigator converts this information into current postures. The navigator then feeds a current posture to the guidance system each ΔT where, the calculation of a new spatial posture is performed.

The navigator module on the simulator has the luxury of knowing the exact vehicle position without the aide of any sensor information. The module simulates receiving sensor data and determines exact position in relation to the specified origin. Also present within this module is a method of introducing a disrupting force so the current posture and the reference path may be significantly different from each other.

3. Autopilot

The autopilot's main function is to receive course correction information from the guidance system and convert this information into electrical signals that will cause movements in the planes and propellers of the vehicle. In the simulator, the product of the autopilot is loaded in the H-matrix [JUREWICZ 90] of the graphics pipeline where the

vehicle's position in relation to the screen is changed [JUREWICZ 90]. Further, there is a corresponding change in planes and propellers on the screen to aid in the visualization of a maneuver. Depending upon the type of guidance system employed by an autonomous vehicle, the information received by the autopilot could take on different forms.

If the cross track guidance method is used, the autopilot will receive rate information from the guidance system. The Yamabico-11 mobile land robot utilizes cross track guidance and therefore serves as a good example of a guidance system that sends rate information to the autopilot.

The autopilot used with the NPS AUV Simulator receives a spatial posture from the guidance system. As previously noted, a spatial posture holds the position, orientation, and curvature information of the vehicle. A series of spatial postures are used as waypoints between the current posture of the vehicle and the desired path of the vehicle. The autopilot will then direct a vehicle from its current position to the next spatial posture to try and achieve the reference path.

4. Obstacle Avoider

The obstacle avoider's main function is to override the reference path specified by the user in the event of encountering an unexpected obstacle. Since the user may not be aware of the obstacle information stored in the vehicle's environmental database, there is great potential of encountering unplanned for obstacles during the course of a mission. In the event that the vehicle's sensors detect an obstacle in the planned path of the vehicle (see [FLOYD 91a]), the obstacle avoider generates emergency waypoint information to navigate past the obstacle. The linking of a series of these waypoints are then used instead of the reference path until safely past the obstacle and until the vehicle returns back to the reference path. Another possible way to navigate around an obstacle would be to use a circular path as described in Figure 3-2. The center of the obstacle may be used as the

definition point of this circular path and the radius value will depend upon the size of the obstruction. The actual method used would be determined within the algorithm that functions as the obstacle avoider.

C. DATA BASES

There are two data stores on Figure 3-1. These two are described below.

1. Mission Log

The mission log is a database that holds current posture information recorded by the navigator. This data store receives a new current posture on each cycle of the control loop. By holding the current posture information, a mission may be saved and replayed at a later time for evaluation purposes. Both the simulator and the testbed vehicle utilize this database for this purpose. This data store also allows testbed vehicle runs to be replayed on the simulator.

2. Environment Models

Before obstacle recognition and positional updating may take place, a suitable environment must be defined. This environment model must facilitate the three functions of path planning, positional identification, and model updating. All three of these functions require the expression of the environment in some type of numerical form. Therefore, linear features are defined by a Cartesian coordinate system where some predefined point serves as the origin. In this manner, all features may be expressed in terms of their x-, y-, and z-coordinates where the linking of three or more vertices defines a polyhedron.

The structure used to link the points of a polyhedron is a linked list. This data structure is allocated in memory containing the x-, y-, and z-coordinates of each point. Pointers are then used to specify which vertices are connected to form the surfaces of a polyhedron.

For the AUV simulator, three different environment models have been developed and are described below.

a. NPS Swimming Pool

The NPS swimming pool serves as one environment for the operation of the AUV simulator. Since the testbed vehicle runs all of its tests in the NPS swimming pool, the graphical representation of this environment is essential. The test runs that will be shown in Chapter VI are run within this particular environment.

The pool has the dimensions of 35.84 meters by 18.40 meters with depths ranging from 1.07 meters at the most shallow point to 2.29 meters at the deepest point. The pool bottom consists of two parts that are of roughly equal size. The first is a flat surface that is of the maximum depth of the pool. The second is a sloping section that begins at 1.07 meters and reaches the maximum depth of 2.29 meters. See Figure 3-4 for pool dimensions.

The pool is graphically represented by using an array structure that holds the dimensions of the six polygons that make up the pool (four walls and two floor surfaces). In addition, a user is able to enter additional objects into the pool environment to facilitate the testing of the guidance mission as well as other missions that are essential to the accurate evaluation of vehicle performance.

b. Monterey Bay

The Monterey Bay has been graphically represented for use in the simulation. A 49 kilometer by 44.4 kilometer rectangle of ocean floor and terrestrial terrain has been digitized in 200 meter increments [JUREWICZ 90]. This area includes the coastline that stretches from the Santa Cruz area in the north to the Pebble Beach area in the south. The

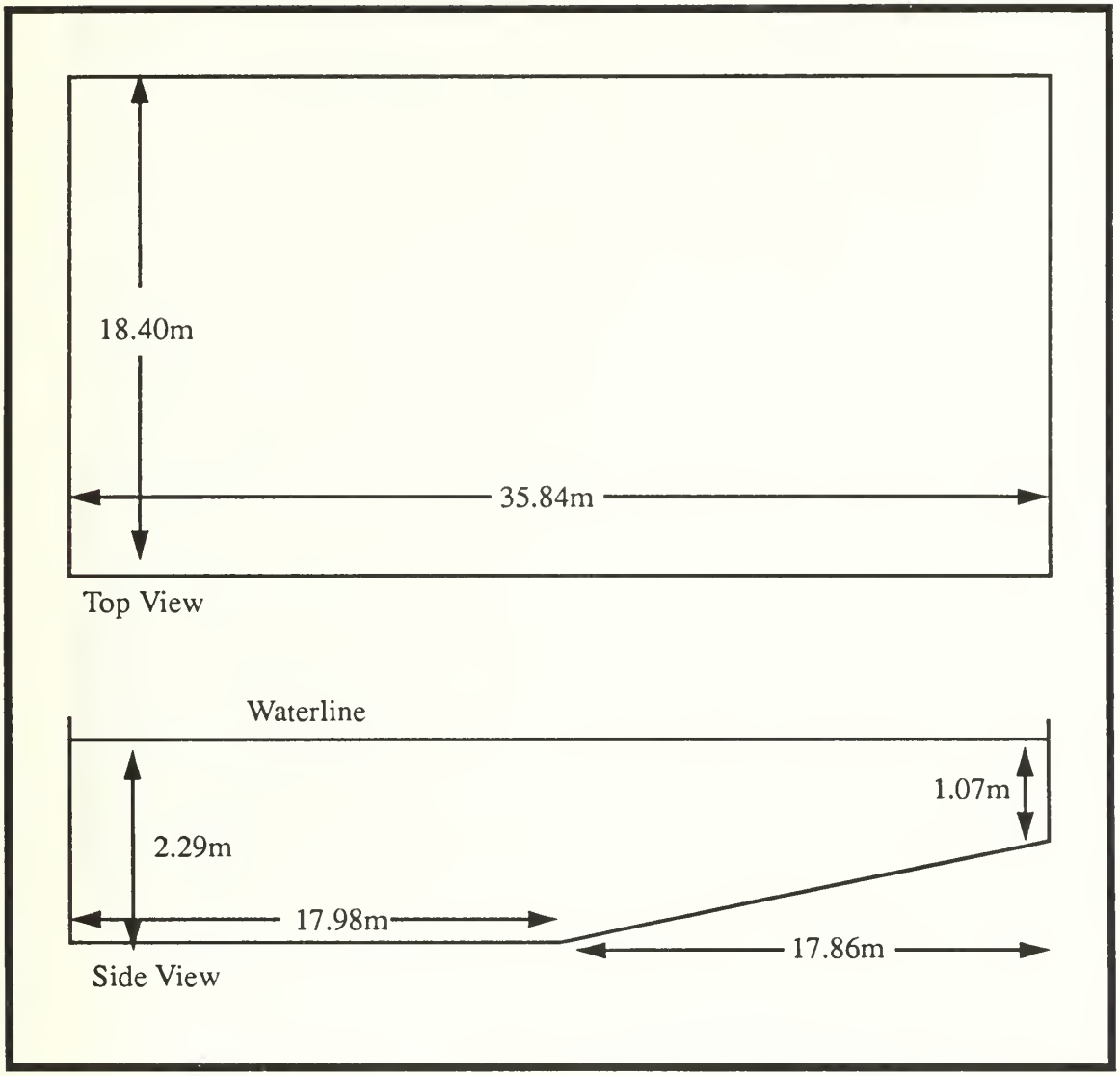

Figure 3-4 **NPS Swimming Pool**

environment has depths that range from zero at the shoreline and includes the Monterey Canyon where depths exceed 7000 feet. See Figure 3-5 for a computer generated aerial view of the bay area.

Due to the spacing of 200 meters between data points, this environment is not especially valuable in the testing and evaluation of the components of the control system. However, this model continues to be invaluable for its original purpose of testing the

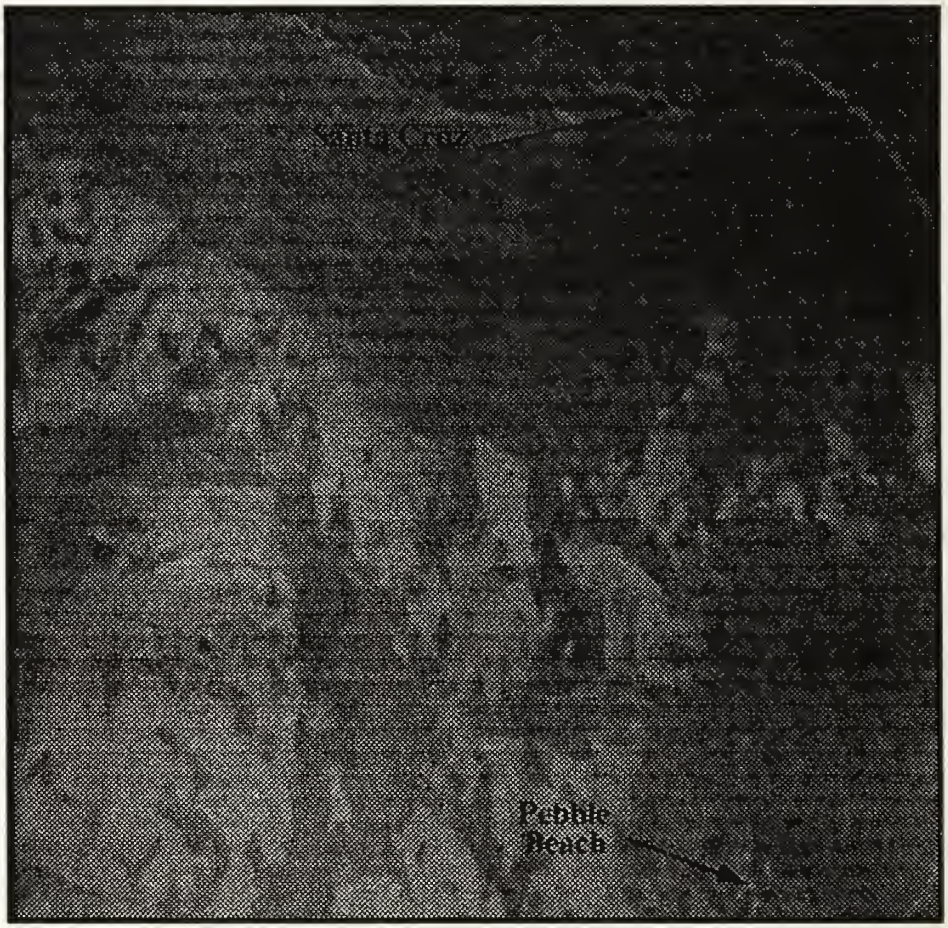

Figure 3-5 **Aerial View of Monterey Bay**

simulator equations of motion and hydrodynamic coefficients in an expansive, real-time setting.

c. Monterey Harbor Boat Basin

The Monterey Harbor boat basin area consists of a 1800 foot by 1440 foot rectangle of ocean floor and terrestrial terrain that has been digitized in 30 foot increments. This area includes the coastline that stretches from Municipal Wharf Two in the south to the U.S. Coast Guard Pier to the north. The depth of the terrain varies from zero feet along the coastline to a maximum depth of 45 feet in the extreme northeast corner of the rectangle. See Figure 3-6 for a computer generated aerial view of the harbor area.

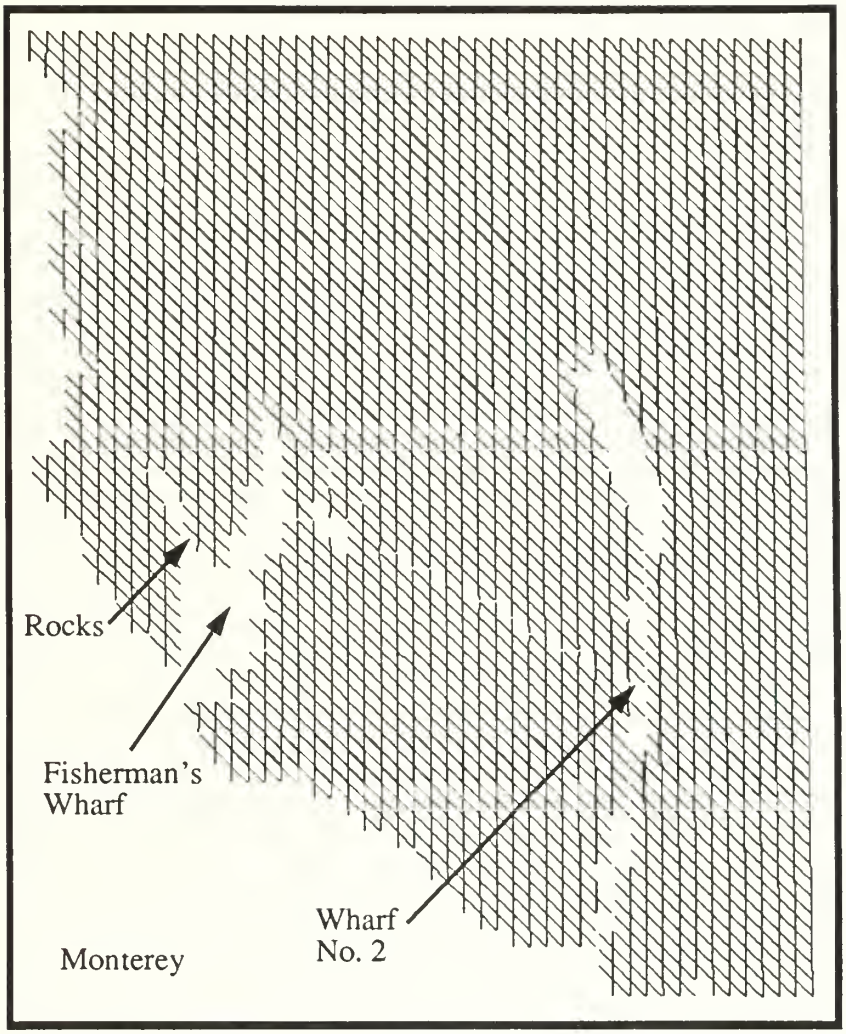

Figure 3-6 **Aerial View of Monterey Harbor**

This view shows the grid pattern of 30 foot by 30 foot terrain.

This model was created so as to provide a suitable ocean environment for the testing of the guidance system, the navigator, the autopilot, and the obstacle avoider. In addition, the linear feature extraction algorithm proposed by Floyd [FLOYD 91a] will be added into this model. The 30 foot spacing between points allows for adequate terrain variation and irregularities which is essential for the testing of the components listed above as well as the feature extraction algorithm.

IV. NPS AUV III SYSTEM

The AUV III Simulator as described in Chapter III does not stand alone. As previously stated, the simulator is modelled after an actual testbed vehicle that is currently operated by NPS. The simulator and the testbed vehicle together form the AUV III System. Within this chapter, the specifications of the testbed vehicle are presented. The vehicle architecture and performance characteristics are then compared with the simulator version.

A. TESTBED VEHICLE DESCRIPTION

1. Specifications

The AUV testbed vehicle as shown in Figure 4-1 is 93 inches long. Not including the length of the planes and rudders, the main body of the vehicle has a 16 inch beam and a height of 10 inches. The total overall displacement of the vehicle is approximately 387 pounds.

The vehicle has four independently controllable planes mounted on the side hull. Two are mounted port and starboard forward and the other two are mounted port and starboard aft. Symmetric with the placement of the planes, the vehicle also has four independently controllable rudders mounted on the top and bottom surfaces of the hull where two are placed forward and two are placed aft (see Figure 4-1).

The propulsion system consists of two aft mounted screws and four tunnel thrusters. The thrusters allow the vehicle to move laterally as well as vertically independent of actual vehicle heading. The presence of these thrusters enables motion in all six degrees of freedom when travelling at higher speeds.

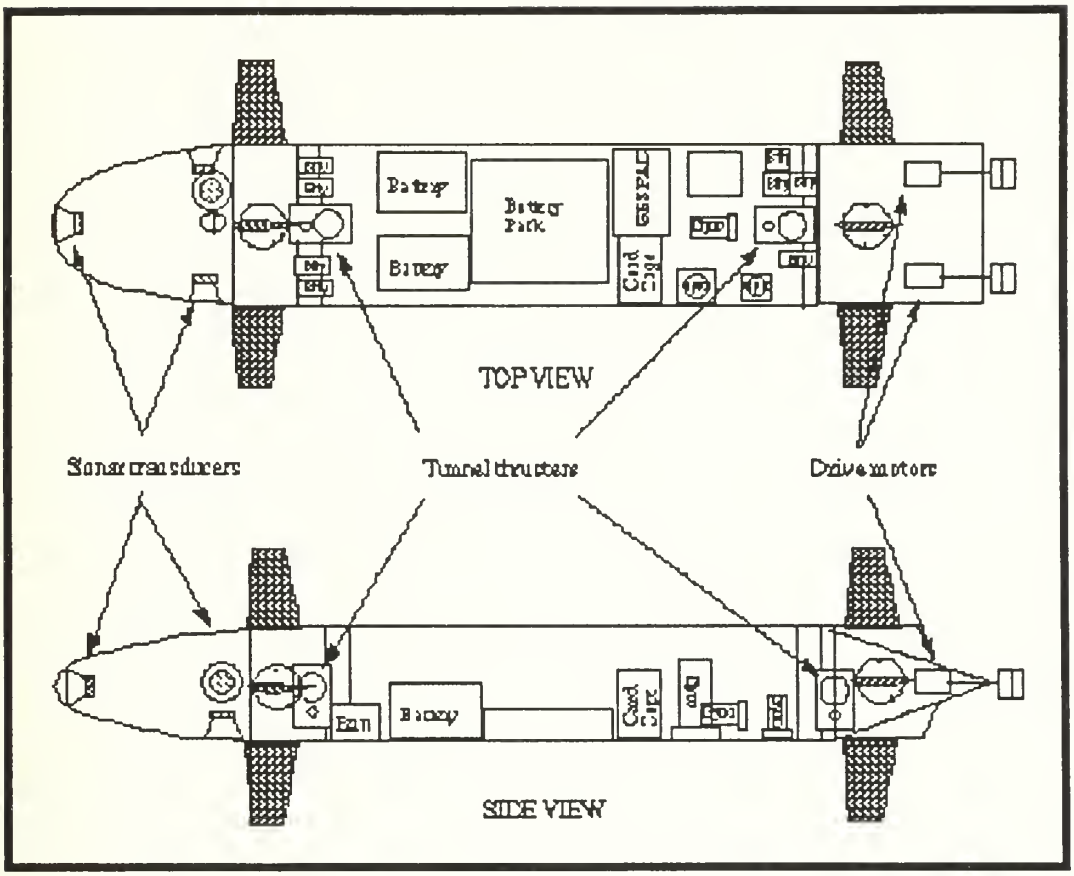

Figure 4-1 NPS AUV III

2. Holonomic Nature of the AUV

By definition, a vehicle that is holonomic is equipped with actuators that will enable it to independently move in as many degrees of freedom as it possesses. Conversely, a non-holonomic vehicle is a vehicle that's movement is restricted in one or more of its degrees of freedom. As mentioned above, the NPS AUV possesses six degrees of freedom. Furthermore, unlike most propellered submersibles, the NPS AUV is considered holonomic.

Vehicles such as the conventional submersible described by Savant [SAVANT 90] are usually non-holonomic. These vehicles possess one or more stern mounted

propellers to move the vehicle along the longitudinal axis (the x component of the posture) and a series of planes to configure the vehicle's orientation (the ϕ, θ, and ψ components of the posture). However, there is no capability to independently move either laterally (the y component of the posture) or vertically (the z component of the posture). Obviously, by utilizing a combination of the four controllable components, movements in the lateral and vertical directions may be realized. However, this movement may entail unacceptable movements in the degrees of freedom for the components used to produce this movement. For example, a non-holonomic vehicle may obtain a posture that is different from its originating posture in the lateral component by adjusting the x and ψ components of its originating posture to arrive at the destination posture. However, the resulting path would be lengthy and may not be the most desirable. A more desirable path may be to move laterally.

The NPS AUV achieves holonomic characteristics through the use of lateral and vertical thrusters. In the above example, the AUV could achieve a lateral displacement by independently changing the lateral component of its six degrees of freedom. Likewise, the vertical thrusters allow the vehicle to change its vertical component independent of the other five components.

B. SIMULATOR/VEHICLE COMPARISON

As stated above, the simulator is designed to model the actual AUV. This goal has, for the most part, been realized. However, there are certain aspects between the two that are significantly different. The following section will discuss these differences and bring to light limitations of the AUV Simulator.

1. Components

As should be apparent by the examination of Figures 3-1 and 4-2, the architectural differences between the testbed vehicle and the simulator model are significant.

In both data flow diagrams, the external processes labelled as OPERATOR signify the human operator of the system. This is where the similarity ends. With the testbed vehicle, the user will have a well defined interface where selections will be made as to mission type and duration. This information will be sent to the PLAN/REPLAN MISSION process where a path will be generated to carry out the mission. The path will then be sent to the EXECUTE MISSION process where the actual reference waypoints will be generated. Since processes 1 and 2 of Figure 4-2 are still in the developmental stage, a different approach was necessary for use with the simulator. Keeping with the spirit of the actual vehicle design, the solution was to bypass the high level user interface and for the user to enter the path in directly. This is accomplished by the user loading a series of waypoints directly into the simulator environment. The waypoints consist of x, y and z position. These waypoints are then processed directly by the guidance system of Figure 3-1 to generate spatial postures.

Another major difference between the architecture of the two is summed up on Figure 3-1 by the external process labelled as SIMULATOR SYSTEMS. In the simulator version, the vehicle systems are incorporated into this one external process. The vehicle systems are those systems that move the vehicle such as the props, planes and thrusters or components such as the sonar equipment. These systems have been lumped into one external process because they are handled differently within the simulator environment then they are handled on the testbed vehicle. On the testbed vehicle, the output of the autopilot is an electrical signal that causes the planes to move to a certain angle and the props to turn at a certain r.p.m. However, in the simulator, the autopilot signals are quite

Figure 4-2 **AUV Vehicle Data Flow Diagram**

different. The autopilot in this case, loads values into the simulator H-matrix [JUREWICZ 90]. The resulting combination of values in this matrix are what cause the control surfaces of the simulator to move and the position of the simulator to be updated.

The remainder of the components on Figure 3-1 function as their counterparts do on Figure 4-2.

2. Performance

Currently, there is insufficient data available to compare the performance of the simulator with the performance of the testbed vehicle. As stated by Jurewicz [JUREWICZ 90], the equations of motion that the simulator uses to reproduce vehicle responses to hydrodynamic forces were based upon data derived from another submersible vehicle. These equations of motion were modified to take into account the characteristics of the AUV, but the equations have yet to be tested against testbed vehicle characteristics.

However, this problem had been planned for by Jurewicz when he included the coefficients package within the simulator software. This package enables the user to adjust the coefficients for the hydrodynamic equations of motion "on the fly" to obtain required results. When testbed vehicle data does become available, all that will be required to obtain correct simulator performance will be to adjust the appropriate coefficients. These new coefficients may then be saved for use as baseline values.

The coefficients package also enables the added advantage of planning for casualty situations. With this package, it is possible for the user to test the resulting vehicle response to control surface hardovers or the response to damage to rudders and planes by introducing adjustments to the coefficients.

V. GUIDANCE SYSTEM

In this chapter, we shall present two methods of guidance. After each is explained in detail, the chapter will conclude with a comparison of test runs conducted using each method.

A. CROSS TRACK GUIDANCE

1. Problem Statement

The cross track guidance scheme is a method of navigation that computes rate information to return to a reference posture. The result of the comparison between current and reference postures are used to calculate an error posture. These error postures are then used to derive target linear and angular velocities. The velocities are then sent to the autopilot which must convert this information into plane and propeller action that will enable a vehicle to return to a specific reference posture.

This system is implemented inside a control loop such as depicted in Figure 5-1. Each iteration of the loop would occur at a specified time interval, Δt. At each Δt, a new reference posture is provided to the guidance system by the path planner and a new current posture is provided to the guidance system by the navigator. The current posture remains as defined in Chapter III. The reference posture, like the current posture, is also a data structure that holds position and orientation information. It is made up of the same six elements from equation 3-1. However, instead of describing the vehicle's actual position, the reference posture specifies the desired position and orientation. Since the goal of the cross track guidance scheme is to arrive at a specified point at a specified time, the guidance system needs a point to achieve, not just a path to obtain.

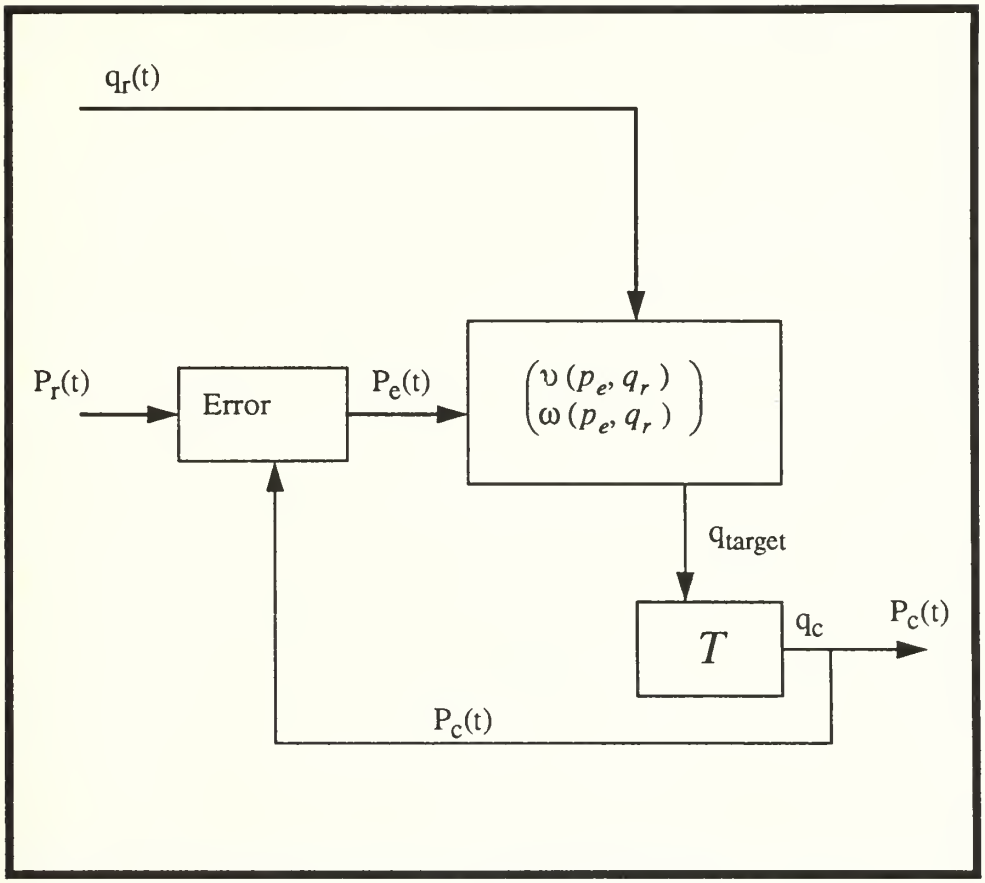

Figure 5-1 **Cross Track Guidance Control Loop**

Error postures are internal to the guidance system and are the difference between the reference posture and the current posture at a time, T. Error postures are expressed in body coordinates. Refer to Figure 5-2 for a two dimensional graphical description of this concept. On this Figure, P_r is the reference posture, P_c is the current posture, and the three components labelled with subscripts e are the derived components of the error posture. The error postures are used within the guidance system to compute target velocities. This concept shall be discussed in detail later in this chapter.

2. Liapunov Stability

Stability is the measurement of the reaction of a vehicle when a change in operating conditions occurs. If the change has little or no effect upon the vehicle, the

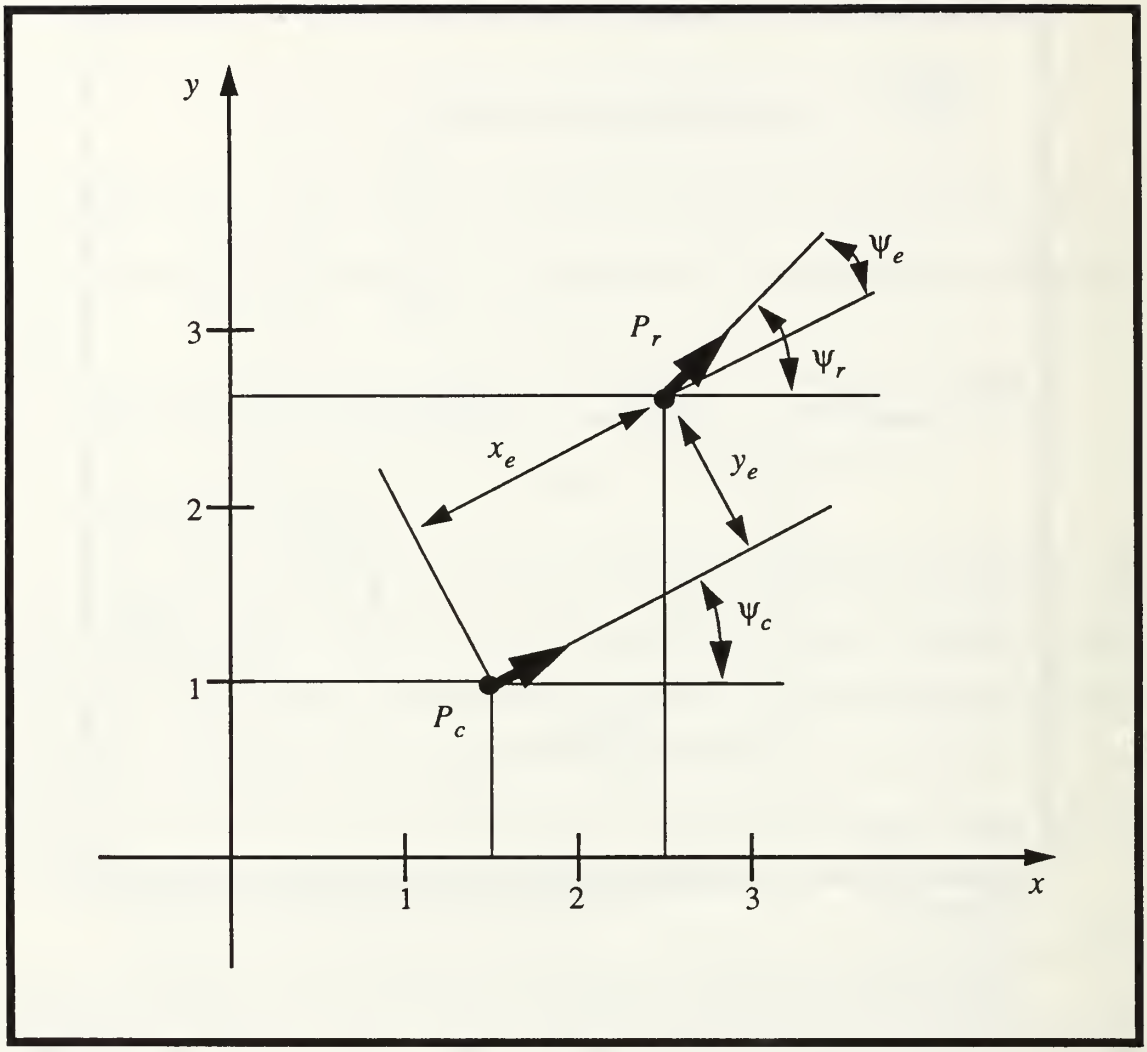

Figure 5-2 **Error Posture Representation**

vehicle remains close to the equilibrium state and is said to be stable. If the vehicle is effected, the vehicle moves away from the equilibrium state and is said to be unstable. In speaking of stability in autonomous systems, we may further break down a stable condition into stable and asymptotically stable. Although stable and asymptotically stable vehicles remain within the equilibrium state, an asymptotically stable vehicle will converge towards the origin. Figure 5-3 gives a graphical description of the three different states. To remain

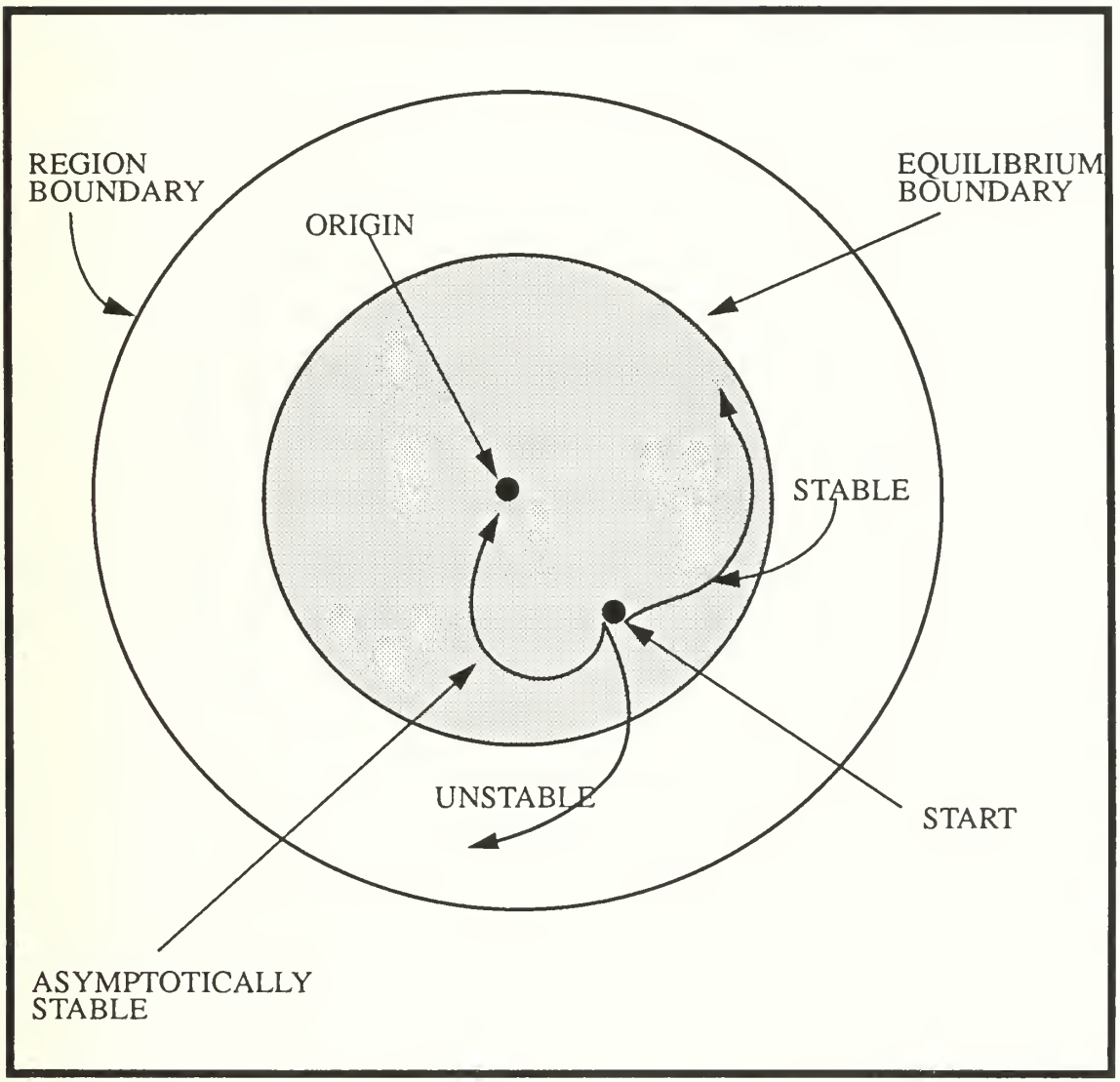

Figure 5-3 **Liapunov Stability**

within equilibrium, a body must not cross the boundary for acceptable error. In this context, acceptable error is dependent upon the design considerations of a vehicle.

The theorems of Liapunov aim to reduce the properties discussed above to a single scaler function, $V(x)$. This function must conform to the following properties:

* $V(x)$ is continuous together with its first partial derivatives in a certain open region about the origin. The outer circle on Figure 5-3 is the open region in question.
* $V(0) = 0$.
* Outside the origin, $V(x)$ is positive. In other words, V is non-negative and goes to zero at the origin.

When $V(x)$ satisfies all of these conditions, it is said to be *positive definite.* In addition, if the first derivative of V is less than or equal to zero, V is called a Liapunov Function [LASALLE 61].

In cross track guidance, finding a stable control rule is accomplished by the use of a Liapunov Function. In [SAVANT 90], a Liapunov Function candidate for a three dimensional control rule was defined as:

$$V = \frac{1}{2}x_e^T x_e + k^T f(\Theta_e) \qquad \text{(Eq. 5-1)}$$

where x_e is defined as the rotation matrix:

$$R = \begin{bmatrix} \cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi & \cos\psi\sin\theta\cos\phi - \sin\psi\sin\phi \\ \sin\psi\cos\theta & \sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix}$$

(Eq. 5-2)

times the spatial coordinates of the reference posture minus the spatial coordinates of the current posture:

$$x_r = (x_r, y_r, z_r)^T \quad \text{(Eq. 5-3)}$$

$$x_c = (x_c, y_c, z_c)^T \quad \text{(Eq. 5-4)}$$

which is represented as follows:

$$x_e = R^T (x_r - x_c) \quad \text{(Eq. 5-5)}$$

The symbol k represents the gain constants $(k_1, k_2, k_3)^T$ and $f(\Theta_e)$ is defined as:

$$f(\Theta_e) = \begin{bmatrix} 1 - \cos\phi_e \\ 1 - \cos\theta_e \\ 1 - \cos\psi_e \end{bmatrix} \quad \text{(Eq. 5-6)}$$

The derivation of the control rule and the proof of equation 5-1 is presented in [SAVANT 90].

3. Calculation of Error Postures

The guidance scheme receives reference posture information from the path planner and current posture information from the navigator in the form:

$$P_r = (x_r, y_r, z_r, \phi_r, \theta_r, \psi_r) \quad \text{(Eq. 5-7)}$$

$$P_c = (x_c, y_c, z_c, \phi_c, \theta_c, \psi_c) \quad \text{(Eq. 5-8)}$$

Where P_r is the reference posture and P_c is the current posture; $x, y,$ and z are the planar locations on the three dimensional plane; and $\phi, \theta,$ and ψ represent pitch, roll, and yaw angles respectively. Reference and current postures are used to calculate error postures. As previously stated, the error posture is the difference, in body coordinates, between the reference posture and the current posture at a time, T. The error posture is defined by the following equations:

$$x_e = \cos\psi_c \cos\theta_c (x_r - x_c) + \sin\psi_c \cos\theta_c (y_r - y_c) - \sin\theta_c (z_r - z_c) \quad \text{(Eq. 5-9)}$$

$$y_e = \cos\psi_c \sin\theta_c \sin\phi_c - \sin\psi_c \cos\phi_c (x_r - x_c) + \sin\psi_c \sin\theta_c \sin\phi_c + \cos\psi_c \cos\phi_c (y_r - y_c) + \cos\theta_c \sin\phi_c (z_r - z_c) \quad \text{(Eq. 5-10)}$$

$$z_e = \cos\psi_c \sin\theta_c \cos\phi_c + \sin\psi_c \sin\phi_c (x_r - x_c) + \sin\psi_c \sin\theta_c \cos\phi_c - \cos\psi_c \sin\phi_c (y_r - y_c) + \cos\theta_c \cos\phi_c (z_r - z_c) \quad \text{(Eq. 5-11)}$$

$$\phi_e = \phi_r - \phi_c \quad \text{(Eq. 5-12)}$$

$$\theta_e = \theta_r - \theta_c \quad \text{(Eq. 5-13)}$$

$$\psi_e = \psi_r - \psi_c \quad \text{(Eq. 5-14)}$$

The results of these equations are then sent to the feedback control loop. See Figure 5-1 for a description of the control loop.

4. Feedback Control Law

The heart of the cross track guidance system is the feedback control law. In this set of equations, the target velocities are computed that will return the vehicle back to the reference posture. We are interested in two sets of velocities. The first are the linear velocities and they are denoted as follows:

$$\upsilon = (u, v, w)^T \quad \text{(Eq. 5-15)}$$

where,

$$u = u_r + u_r (\cos\psi_e \cos\theta_e - 1) + K_x x_e \quad \text{(Eq. 5-16)}$$

$$v = v_r + v_r(\sin\psi_e\cos\theta_e - 1) + K_x x_e \qquad \text{(Eq. 5-17)}$$

$$r = r_r + r_r(-\sin\theta_e - 1) + K_x x_e \qquad \text{(Eq. 5-18)}$$

The constant K_x in the above equations is a convergence factor that relates to ΔT. The angles and velocities are either marked with subscript *e* or *r* and denote error or reference.

The second velocities are the angular velocities and they are denoted as:

$$\omega = (p, q, r)^T \qquad \text{(Eq. 5-19)}$$

where,

$$p = p_r + q_r(\sin\phi_r\tan\theta_r - \sin\phi_c\tan\theta_c) +$$
$$r_r(\cos\phi_r\tan\theta_r - \cos\phi_c\tan\theta_c) + K_1\sin\phi_e -$$
$$\sin\theta_e\left(\frac{-z_e u_r\sin\theta_e}{K_3\sin\psi_e} + q_r(\sin\phi_r\sec\theta_r - \sin\phi_c\sec\theta_c) +\right.$$
$$r_r(\cos\phi_r\sec\theta_r - \cos\phi_c\sec\theta_c) + K_3\sin\psi_e) \qquad \text{(Eq. 5-20)}$$

$$q = q_r + \cos\phi_e\left(\frac{y_e u_r\cos\theta_e\sin\psi_e}{\sin\theta_e K_2} + q_r(\cos\phi_r - \cos\phi_c) -\right.$$
$$r_r(\sin\phi_r - \sin\phi_c) + K_2\sin\theta_e) +$$
$$\sin\phi_e\cos\theta_e\left(\frac{-z_e u_r\sin\theta_e}{K_3\sin\psi_e} + (q_r(\sin\phi_r\sec\theta_r - \sin\phi_c\sec\theta_c) + \right.$$
$$r_r(\cos\phi_r\sec\theta_r - \cos\phi_c\sec\theta_c) + K_3\sin\psi_e)) \qquad \text{(Eq. 5-21)}$$

$$r = r_r - \sin\phi_e\left(\frac{y_e u_r\cos\theta_e\sin\psi_e}{\sin\theta_e K_2} + q_r(\cos\phi_r - \cos\phi_c) -\right.$$

$$r_r(\sin\phi_r - \sin\phi_c) + K_2\sin\theta_e)) +$$

$$\cos\phi_e\cos\theta_e\left(\frac{-z_e u_r \sin\theta_e}{K_3\sin\psi_e} + q_r(\sin\phi_r\sec\theta_r - \sin\phi_c\sec\theta_c) +\right.$$

$$r_r(\cos\phi_r\sec\theta_r - \cos\phi_c\sec\theta_c) + K_3\sin\psi_e) \qquad \text{(Eq. 5-22)}$$

The constants K_1, K_2, and K_3 relate to convergence. The angles and velocities are either marked with subscripts *c, e,* or *r* and denote current, error, or reference respectively.

The velocities u, v, w, p, q, r describe the speed of the vehicle in all six degrees of freedom. In a typical underwater vehicle, the guidance scheme would only be concerned with the angular velocities and the u component of the linear velocities. This is due to the fact that the propulsion source would be propellers at the rear of the vehicle. However, in the NPS AUV, the v and w components also must be considered due to the presence of vertical and horizontal thrusters.

5. Three Dimensional Test Results

In this section, the results of a test run using cross track guidance are presented. These test runs are applied to a point mass and are independent of AUV vehicle dynamics. In the test run, the initial reference posture is:

$$P_r = (0, 0, 0, 0, 0, 0) \qquad \text{(Eq. 5-23)}$$

and is incremented each Δt by:

$$x_r = x_r + (\Delta t \times u_r) \qquad \text{(Eq. 5-24)}$$

where the reference longitudinal velocity, u_r, is held at a constant value of 0.1 and Δt is a ten hertz cycle. In other words, if a line connected all the reference postures, the postures would appear at equal increments along the x-axis beginning at the origin and increasing in the positive x direction.

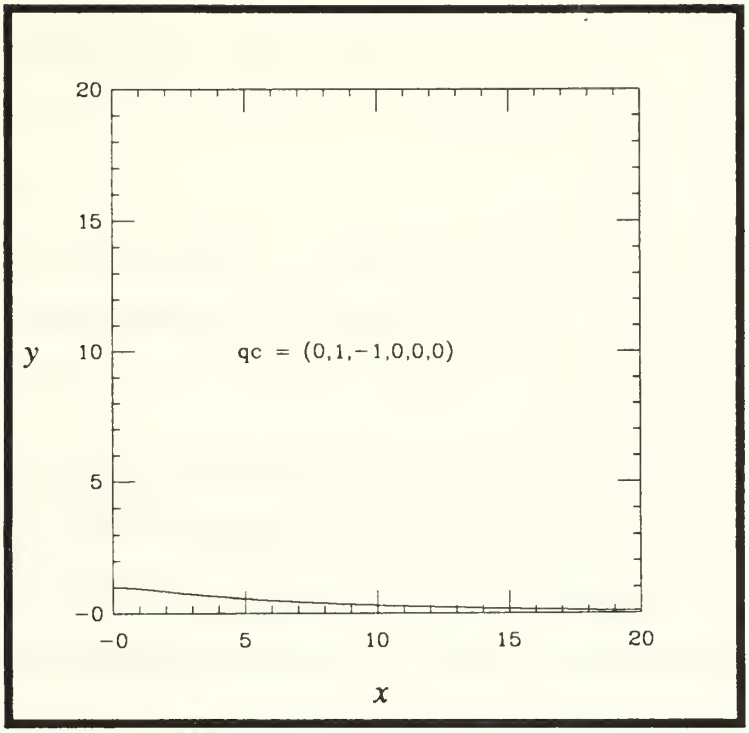

Figure 5-4 Cross Track Guidance Convergence (*x* versus *y*)

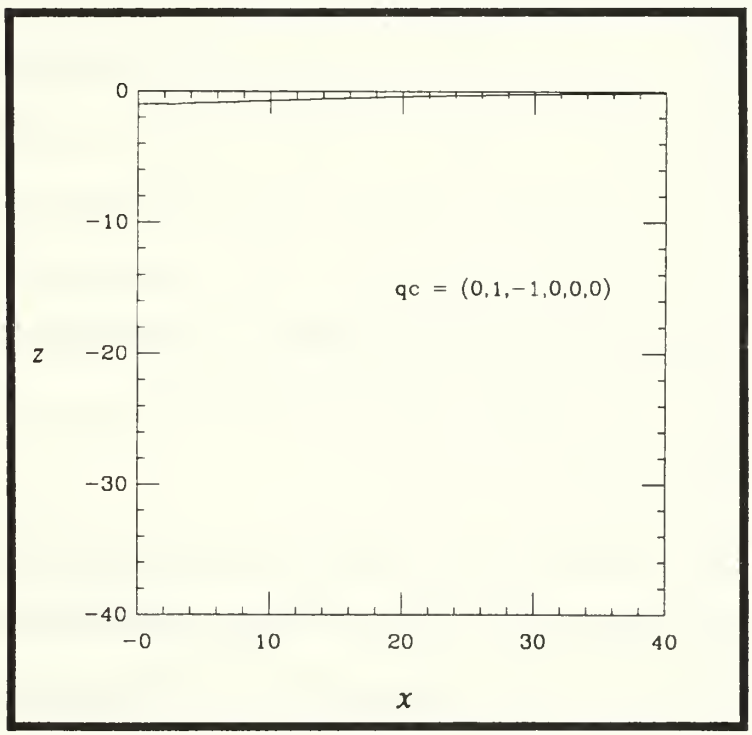

Figure 5-5 Cross Track Guidance Convergence (*x* versus *z*)

Figure 5-4 is a graph of the x versus y position with respect to time. The results reflect the initial parameters entered as:

$$P_c = (0, 1, -1, 0, 0, 0) \quad \text{(Eq. 5-25)}$$

The result of the test run is a smooth convergence towards the reference posture. Figure 5-5 is the graphical view of x versus z for the same initial current posture.

B. SPATIAL TRACKING

The guidance scheme chosen for implementation on the AUV Simulator is spatial tracking. The mechanics of this method will be explained in detail below. Although this method only returns a vehicle back to a reference path and not to a specific posture, it is extremely simple to implement. Additionally, there is no significant increase in complexity when the algorithm is expanded to three dimensions.

For simplicity, the initial discussion of spatial tracking will be discussed using only the x, y plane. Therefore, the postures defined in Chapter III are redefined as:

$$P_c = (x, y, \psi) \quad \text{(Eq. 5-26)}$$

and,

$$P_s = (x, y, \psi, \kappa) \quad \text{(Eq. 5-27)}$$

for the purposes of this discussion. The expanded three dimensional implementation shall be presented later in this section.

1. Problem Statement

Given the initial configuration described by Figure 5-6, it is desirable to find a smooth and continuous path from the current posture, q_c, to the reference path which is defined by the point q_0. As stated in Chapter III and modified above, the current posture, q_c, is defined by:

$$q_c \equiv (x_c, y_c, \Psi_c) \qquad \text{(Eq. 5-28)}$$

and the reference path, q_0 is defined by:

$$q_0 \equiv (x_0, y_0, \Psi_0) \qquad \text{(Eq. 5-29)}$$

Within the example of Figure 5-6, we are assuming a reference path that consists of an infinite directed line that passes through the point specified by x and y, and direction is specified by the angle ψ. The choice of q_0 is arbitrary and will satisfy the requirements of the guidance equations as long as this point is located somewhere along the line that forms the reference path. However, it is convenient for this example to chose q_0 to be relatively close to the current posture, q_c. The point H is the point along the reference path that is closest to the current posture. This point is also called the image point. The point A is a point along the reference path an arbitrary distance s_0, from point H and is the point on the reference path that the vehicle is directed to steer towards. This point is also called the forrunner point. It will be shown later how different values of s_0 effect the rate of convergence towards the reference path. The angle α is the orientation from q_c to A measured from the current heading of the vehicle. The angle β is the orientation from q_c to q_0 minus the perpendicular to the reference path. The distances d, d_0, and d_1 are self explanatory. The goal is to use this information to calculate the required curvature, κ, that the vehicle needs to achieve to return to the reference path.

2. Curvature

In an automobile, the curvature angle would be equivalent to the steering wheel position of the vehicle. While travelling in a straight path, the steering wheel is neutral and the curvature equals zero. If the wheel is turned to the left, curvature becomes greater than

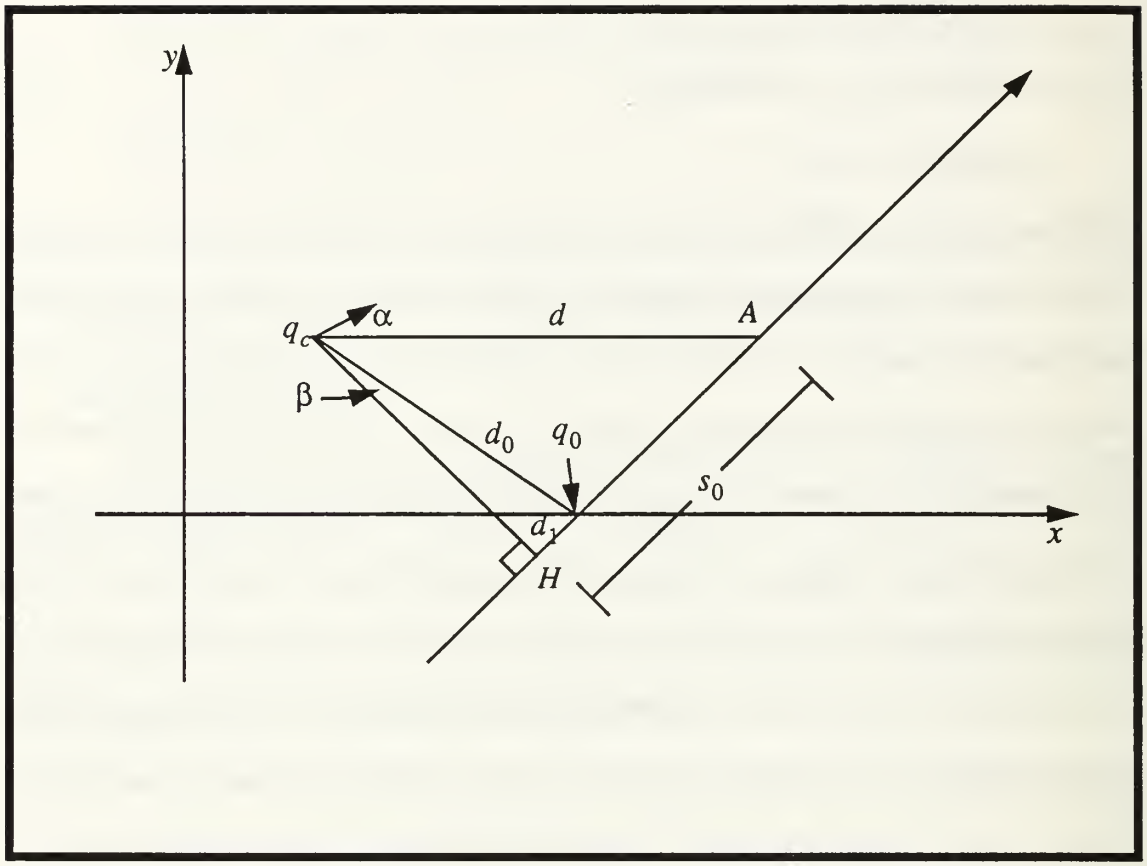

Figure 5-6 **Spatial Situation**

zero while a turn to the right results in a negative curvature. If the wheel is positioned at a constant angle and the vehicle is allowed to traverse a 360 degree turn,

$$\kappa = \frac{1}{r} \qquad \text{(Eq. 5-30)}$$

where r is the radius of the resulting circle.

In the AUV, the steering wheel angle would correspond to the positions of the control surfaces. The cumulative effect of the orientation of the four rudder surfaces is the curvature angle of the AUV within the scope of this two dimensional discussion.

3. Calculation of Curvature

Curvature is calculated using geometric principles. Referring to Figure 5-6, the goal is to have point q_c converge with point A. As defined in equation 5-30, curvature is a function of the radius of the circle. At each time interval, Δt, a particular curvature is defined by the circle connecting q_c with A.

However, at a particular Δt, the only known values of Figure 5-6 are the current posture, q_c, which is received from the navigator and the reference waypoint, q_0, which is supplied by the user specified mission requirements (see Figure 3-1). With these two points, it is possible to calculate the radius of the circle formed by the points q_c and A. There are two separate cases to consider. The first case follows.

As stated above, the image point, H, is the closest point of approach that the current posture is to the reference path at a particular Δt. Therefore, a line drawn from q_c to H is perpendicular to the reference path and the length d_0 may be calculated by:

$$d_0 = \sqrt{(x_0 - x_c)^2 + (y_0 - y_c)^2} \qquad \text{(Eq. 5-31)}$$

Also, the angle β is calculated by:

$$\beta = \operatorname{atan2}\left(\frac{y_0 - y_c}{x_0 - x_c}\right) - (\psi_0 - 90^\circ) \qquad \text{(Eq. 5-32)}$$

The distance $d1$ may then be determined by:

$$d_1 = d_0 \sin\beta \qquad \text{(Eq. 5-33)}$$

which fixes the point H using the following equations:

$$H_x = x_0 - d_1 \cos\psi_0 \qquad \text{(Eq. 5-34)}$$

$$H_y = y_0 - d_1 \sin\psi_0 \qquad \text{(Eq. 5-35)}$$

Once the image point is determined, the forrunner point A is calculated using the convergence factor s_0 in the equations:

$$A_x = H_x + s_0 \cos\psi_0 \qquad \text{(Eq. 5-36)}$$

$$A_y = H_y + s_0 \sin\psi_0 \qquad \text{(Eq. 5-37)}$$

With the fixing of the forrunner point, A, the distance between q_c and A is:

$$d = \sqrt{(A_x - x_c)^2 + (A_y + y_c)^2} \qquad \text{(Eq. 5-38)}$$

and the orientation from q_c to A is calculated by:

$$\alpha = \operatorname{atan2}\left(\frac{A_y - y_c}{A_x - x_c}\right) \qquad \text{(Eq. 5-39)}$$

This makes the calculation of the radius possible using the equation:

$$r = \frac{d}{2\sin(\alpha - \psi_c)} \qquad \text{(Eq. 5-40)}$$

See Figure 5-6 for a graphical representation of equation 5-40.

The above case is sufficient to guide the vehicle to the reference path. However, as depicted in Figure 5-7, if the above method is used exclusively, the resulting path will overshoot the reference path. Our desire is to achieve the reference path smoothly, without oscillation. Therefore, a second case is required that will adjust the curvature at a derived point so as to prevent oscillations and still remain relatively smooth.

First, the threshold point between the two cases must be determined. Referring to Figure 5-8, the sign of the angle of ψ_1 determines which case to use. The angle α is the orientation from q_c to A and is determined by:

$$\alpha = \operatorname{atan2}(A_y - y_c, s_0) \qquad \text{(Eq. 5-41)}$$

The angle β is the difference between ψ_c and α and is expressed as:

$$\beta = \text{normal}(\alpha - \psi_c) \qquad \text{(Eq. 5-42)}$$

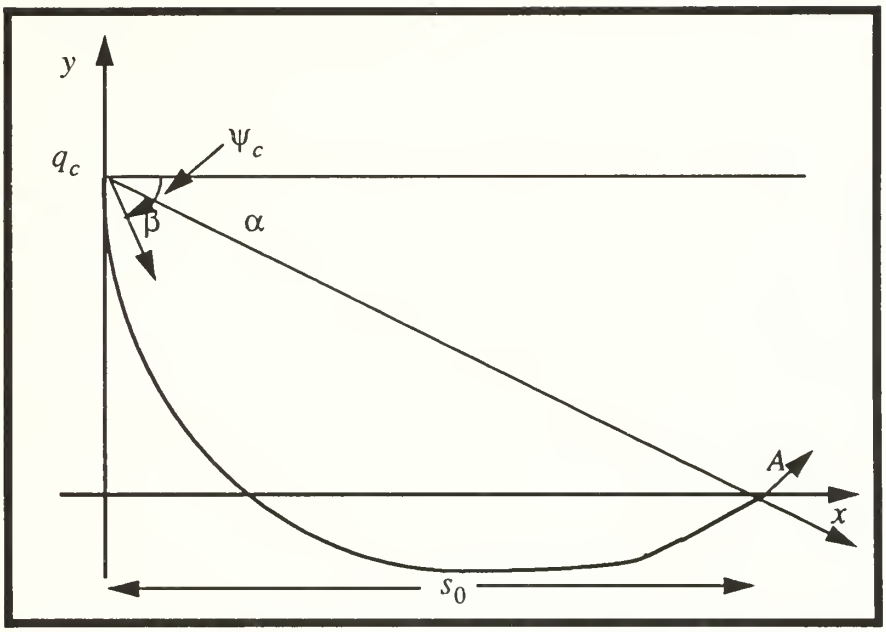

Figure 5-7 Oscillatory Result from the Exclusive Use of Case I

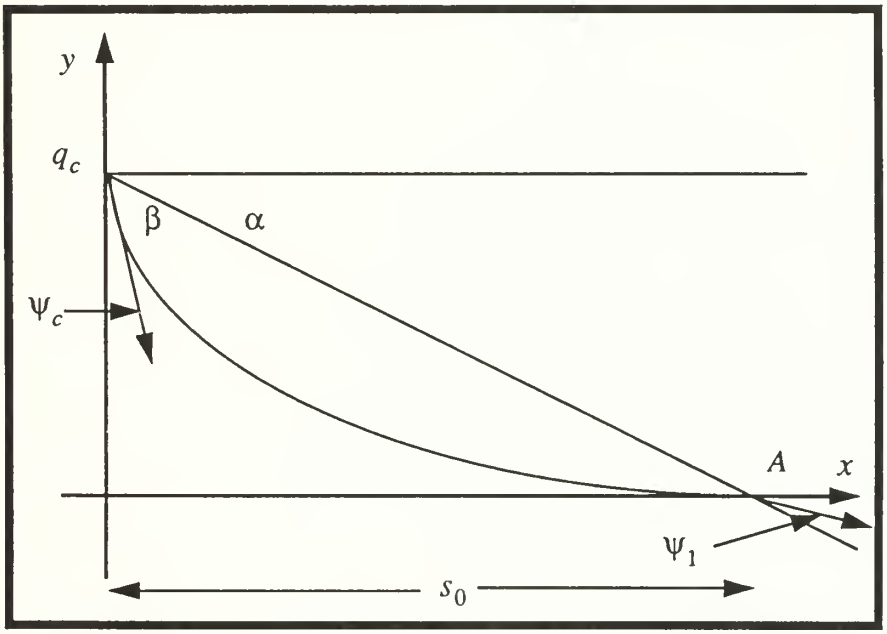

Figure 5-8 Desired Result Using Cases I and II

The angle ψ_1 is then simply:

$$\psi_1 = \alpha + \beta \qquad \text{(Eq. 5-43)}$$

If ψ_1 is negative, the case described above is used. However, when the value of ψ_1 changes from negative to positive, the following equation is used to calculate κ:

$$\kappa = \frac{s_0 (1 - \cos (\psi_c - \psi_p)) ^2}{d_0 \cos\beta \times |d_0 \cos\beta \times \sin (\psi_c - \psi_p)|} \qquad \text{(Eq. 5-44)}$$

where ψ_p is the orientation of the reference path. This equation is derived geometrically as follows. Referring to Figure 5-9, and using the definition of curvature as defined in equation 5-30:

$$y = r - r(\cos\psi) \qquad \text{(Eq. 5-45)}$$

Solving for the radius r, this equation may be rewritten as:

$$r = \frac{d_0 \cos\beta}{1 - \cos\psi} \qquad \text{(Eq. 5-46)}$$

Therefore, the curvature may be determined by:

$$\kappa = \frac{1 - \cos\psi}{d_0 \cos\beta} \qquad \text{(Eq. 5-47)}$$

Experimental results using this equation still produces an overshoot situation. To rectify this problem, the curvature produced by this equation is multiplied by:

$$\frac{s_0}{s_1} \qquad \text{(Eq. 5-48)}$$

where s_1 is the point of intersection with the reference path in an overshoot situation and is specified as:

$$s_1 = \frac{-d_0 \cos\beta \sin (\psi_c - \psi_p)}{1 - \cos\beta} \qquad \text{(Eq. 5-49)}$$

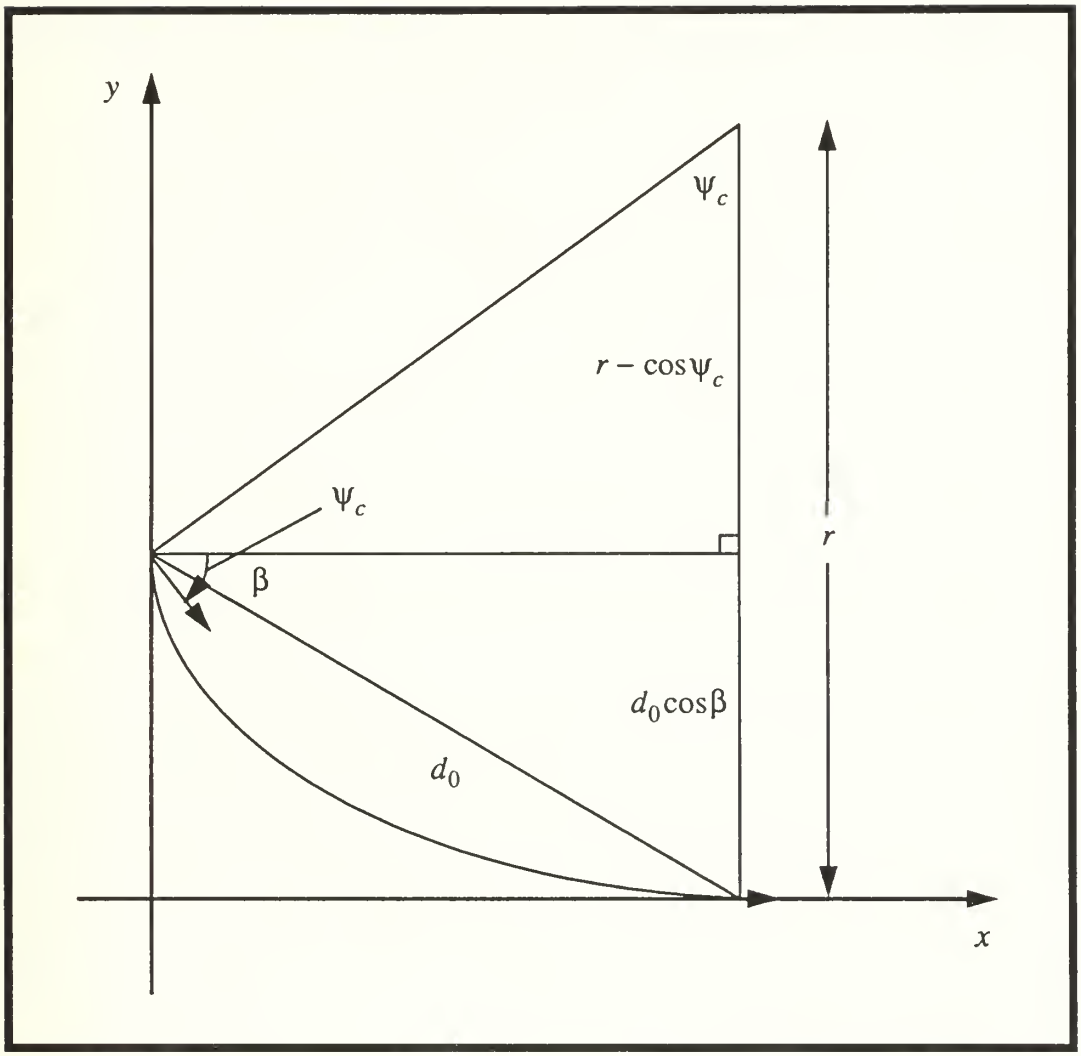

Figure 5-9 **Geometric Situation for Case II**

When this factor is multiplied in, the result is a convergence towards the reference path with no overshoot. However, the use of equation 5-48 is limited to cases where s_1 is positioned between the image point and the forrunner point. If s_1 is outside this interval, equation 5-47 is used without the multiplication factor.

It should be pointed out that the exclusive use of equations 5-44 and 5-47 will not produce the desired results either. The use of these two equations will limit κ to turns in one direction only. This will cause the vehicle to reach the reference path, but with an

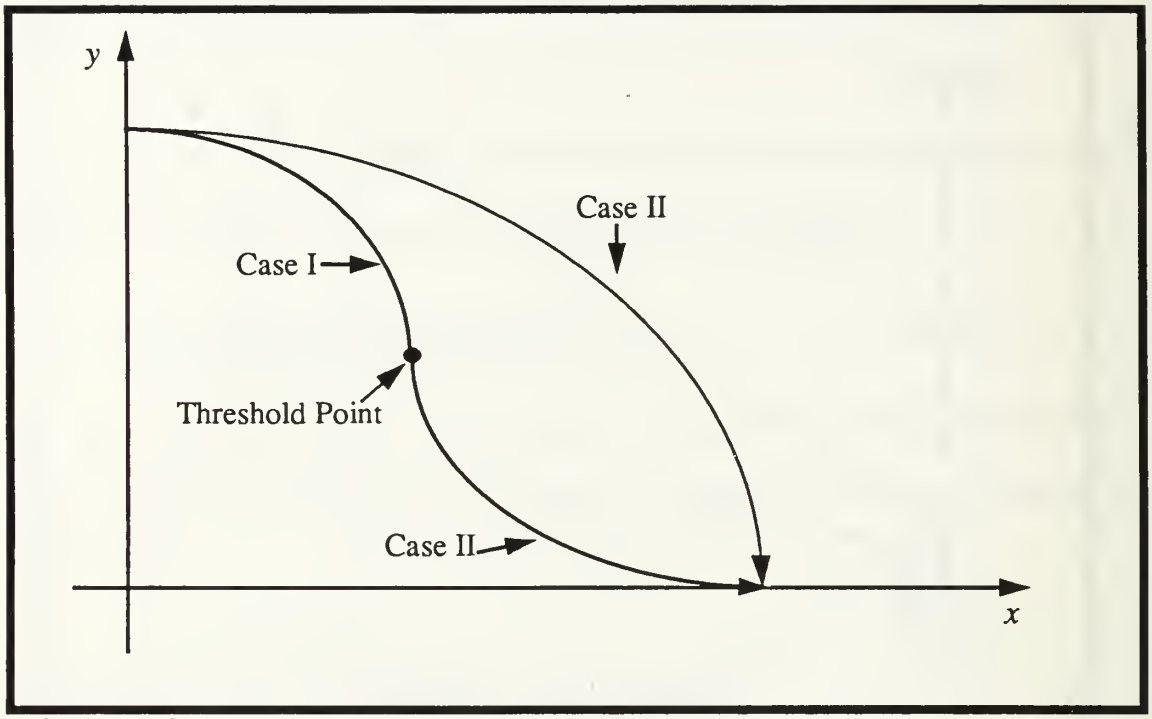

Figure 5-10 **Resulting Path from Exclusive Use of Case II**

undesirable orientation. The threshold point between the use of equations 5-40 and 5-44/5-47 ensures that the vehicle is in proper position so as to reach the reference path with the proper orientation. See Figure 5-10 for a graphical representation.

4. Three Dimensional Spatial Tracking

Up to this point, the discussion of spatial tracking has been presented only in two dimensions. It is our contention that an expansion to three dimensions requires very little additional overhead. As stated earlier, the curvature angle is equated to the positions of the rudder planes in the two dimensional situation. Movement of the rudders causes a corresponding change in the yaw angle of the vehicle in the x, y plane. Along the same lines, the planes of the vehicle enable the vehicle to change its pitch angle in the x, z plane. Therefore, the same equations used to compute the curvature in the x, y plane may be used to compute curvature in the x, z plane. Instead of using y and ψ values in equations 5-31

thru 5-49, we substitute in corresponding values of z and ϕ. The curvature in the x, z plane, as listed in equation 3-2, has been denoted as Γ.

5. Calculation of Spatial Postures

Once the curvature angles have been determined, the last task in each iteration is to update the vehicle's position. This is accomplished using the following equations:

$$x_c = x_c + (\Delta s \cos(\psi_c + \frac{\Delta\psi}{2}) \cos\phi_c) \quad \text{(Eq. 5-50)}$$

$$y_c = y_c + (\Delta s \sin(\psi_c + \frac{\Delta\psi}{2}) \cos\phi_c) \quad \text{(Eq. 5-51)}$$

$$z_c = z_c + (\Delta s \sin\phi_c) \quad \text{(Eq. 5-52)}$$

$$\phi_c = \phi_c + (\Delta s \Gamma) \quad \text{(Eq. 5-53)}$$

$$\psi_c = \psi_c + (\Delta s \kappa) \quad \text{(Eq. 5-54)}$$

where Δs is:

$$\Delta s = \Delta t u_r \quad \text{(Eq. 5-55)}$$

and $\Delta\psi$ is:

$$\Delta\psi = \Delta s \kappa \quad \text{(Eq. 5-56)}$$

The resulting spatial posture is then sent on to the autopilot. See Figure 3-1.

6. Three Dimensional Test Results

Using the same test parameters used to show the results of cross track guidance (Eq. 5-25), a test was run with the spatial tracking algorithm. Figures 5-11 and 5-12 are the graphs of the results in the x, y plane and x, z planes respectively. On each, the result is a smooth convergence towards the x-axis.

Figure 5-13 is a graph showing the different results from one planar position to the reference path using several different values for ψ_c. The first initial heading is specified

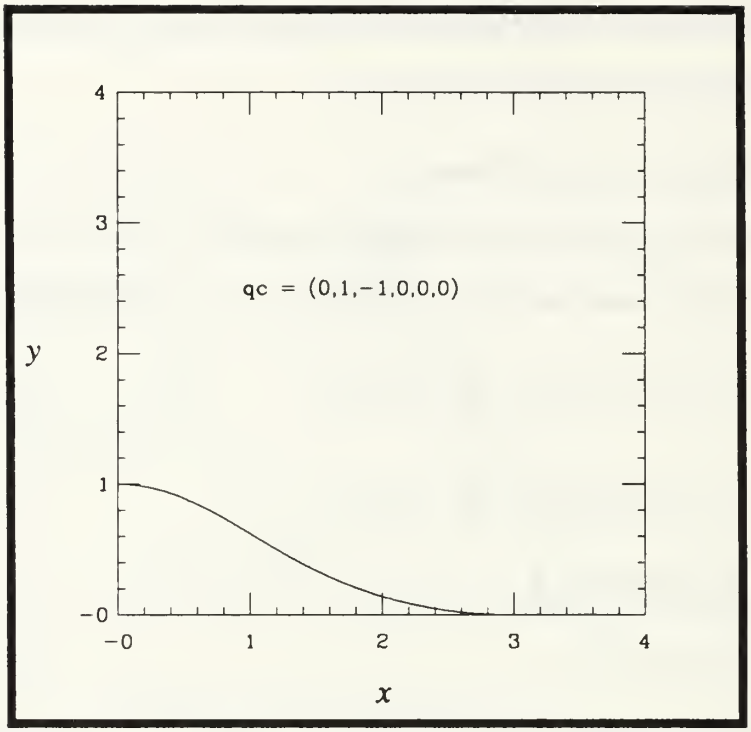

Figure 5-11 Spatial Tracking Convergence (x versus y)

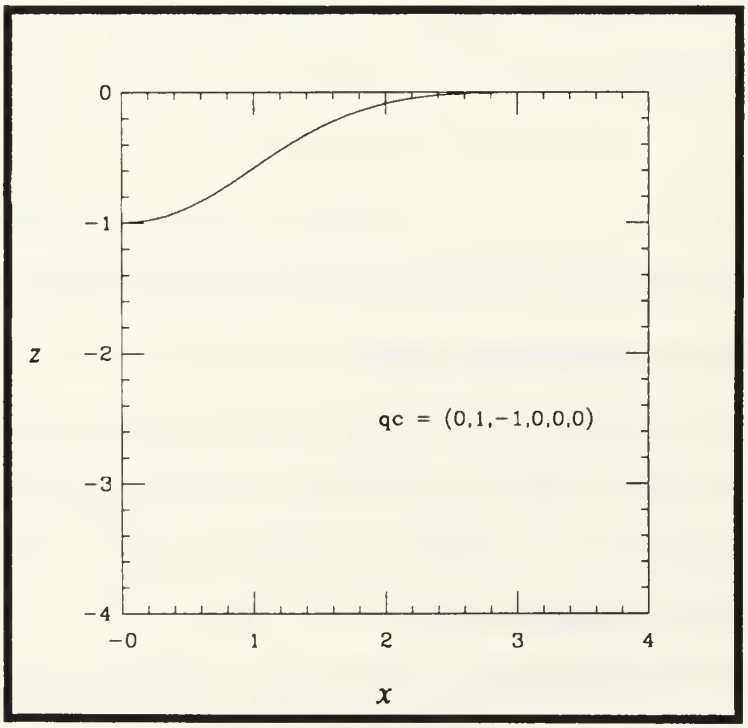

Figure 5-12 Spatial Tracking Convergence (x versus z)

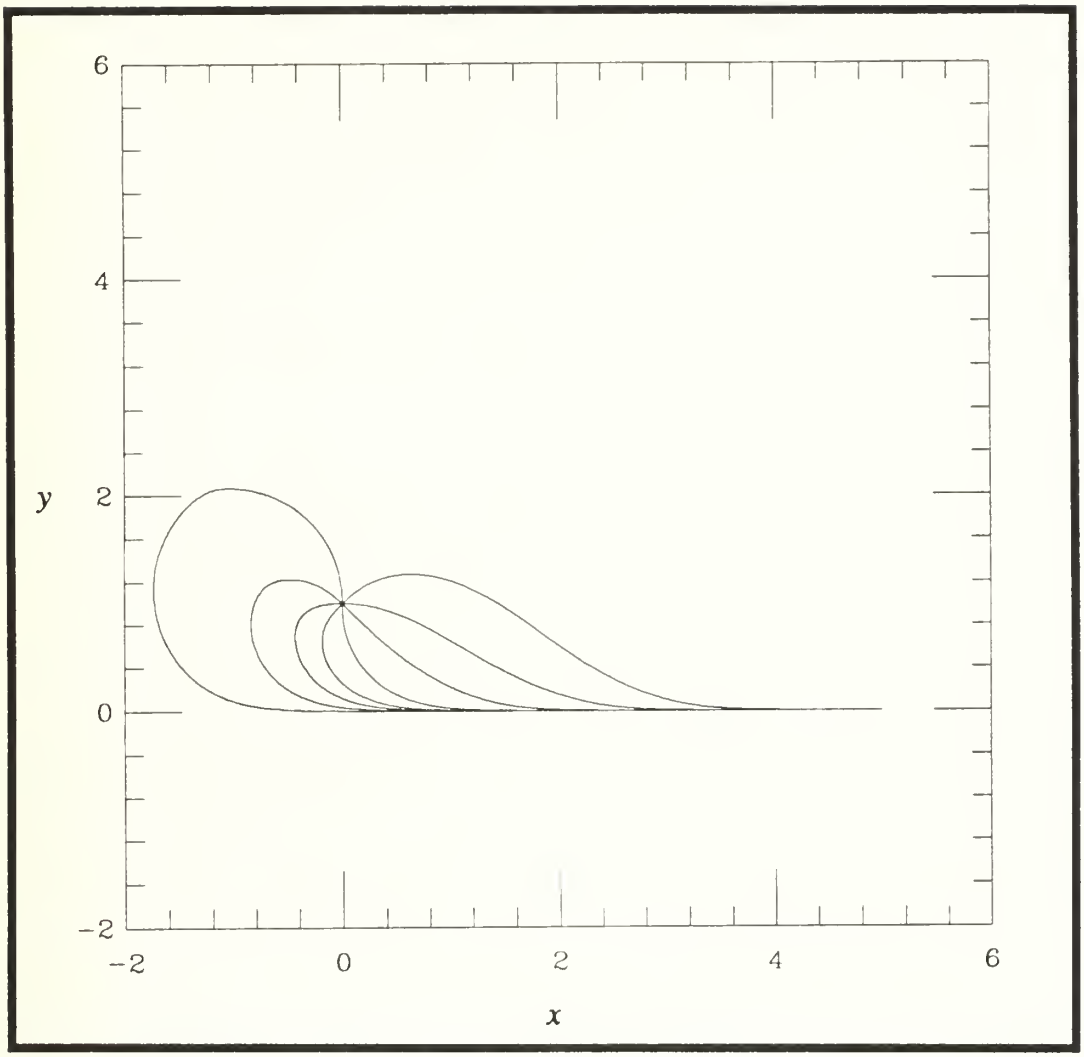

Figure 5-13 **Convergence from any Initial Heading**

as zero and each successive heading is oriented $\frac{\pi}{4}$ from the previous. As depicted, the algorithm gives a smooth convergence from any initial orientation to the reference path.

7. Convergence Characteristics

Referring to Figure 5-6, the distance between the image point, H, and the forrunner point, A, has been labelled as s_0. As previously stated, the forrunner point is the point that the vehicle is directed to steer towards at a particular Δt. As the length of s_0 is

changed, the space required for convergence is directly proportional while the value of the curvature is inversely proportional. In other words, as s_0 increases, space required for convergence increases and the curvature value decreases.

Figure 5-14 shows the different space requirements for convergence using different values of s_0. Figure 5-15 uses the same values of s_0, but shows their effect on κ. Notice the trade off between the value of κ and the space required for convergence.

C. IMPLEMENTATION COMPARISON

In the following section, the cross track and the spatial tracking algorithms are compared. Figure 5-16 is the graph of both methods in regards to space required for convergence.

1. Cross Track Guidance

One advantage of this method is that the scheme allows the vehicle to return to the desired reference posture with respect to time. In other methods of guidance, an acceleration scheme is required if this result was desirable. Additionally, this method has been proven to be stable by the use of Liapunov functions [KANAYAMA 90]. This system produced cubic spirals as the desired path to follow between postures which had the effect of minimizing jerk between waypoints. The minimization of jerk is a very desirable feature for a guidance system of an underwater vehicle.

The major disadvantage of this scheme is complexity. Although the scheme has been successfully implemented on the Yamabico-11 mobile land robot [KANAYAMA 90], three dimensional applications become unnecessarily complex when compared with other guidance methods. A key to the derivation of the target velocities are the gains required for critically damped convergence. In [KANAYAMA 90], a relation was proposed and proved that linked the two gains necessary to compute rotational velocity. In the three dimensional

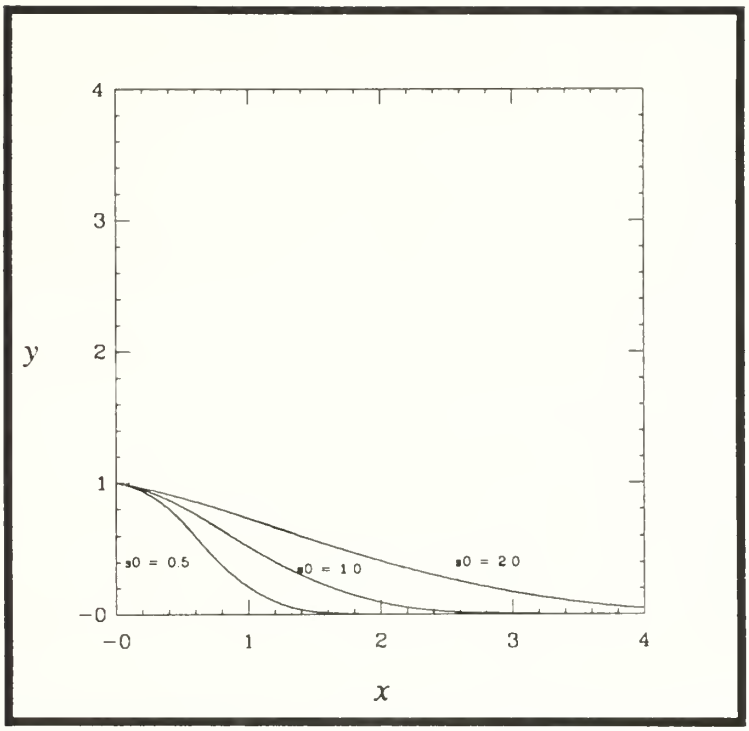

Figure 5-14 Effects of s_0 on Space Required for Convergence

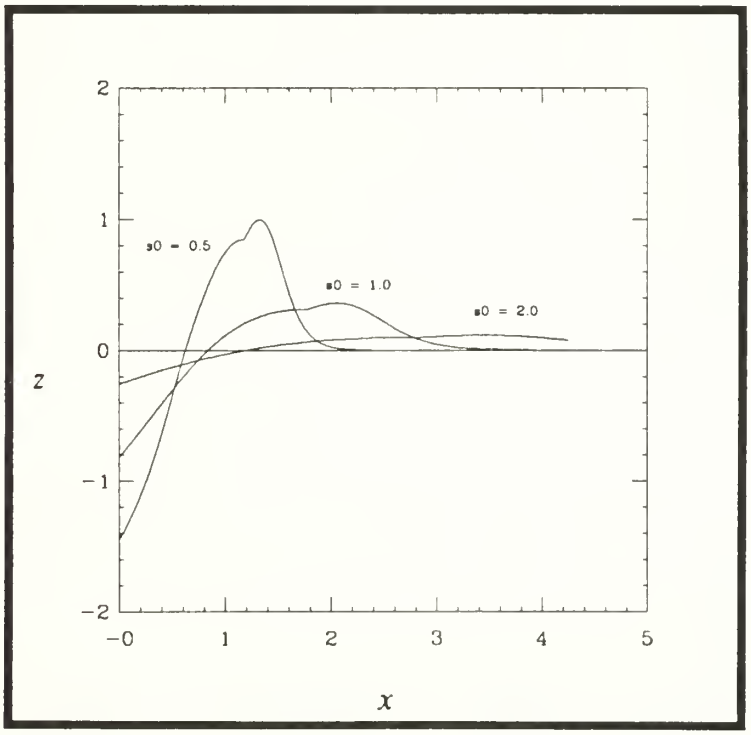

Figure 5-15 Effects of s_0 on κ

application of this method, this relation must still hold. However, instead of one target rotational velocity computation, the three dimensional application requires target velocity for pitch, roll, and yaw. Additionally, the equations that produce these target velocities are computatively expensive which limit the performance of a real-time system.

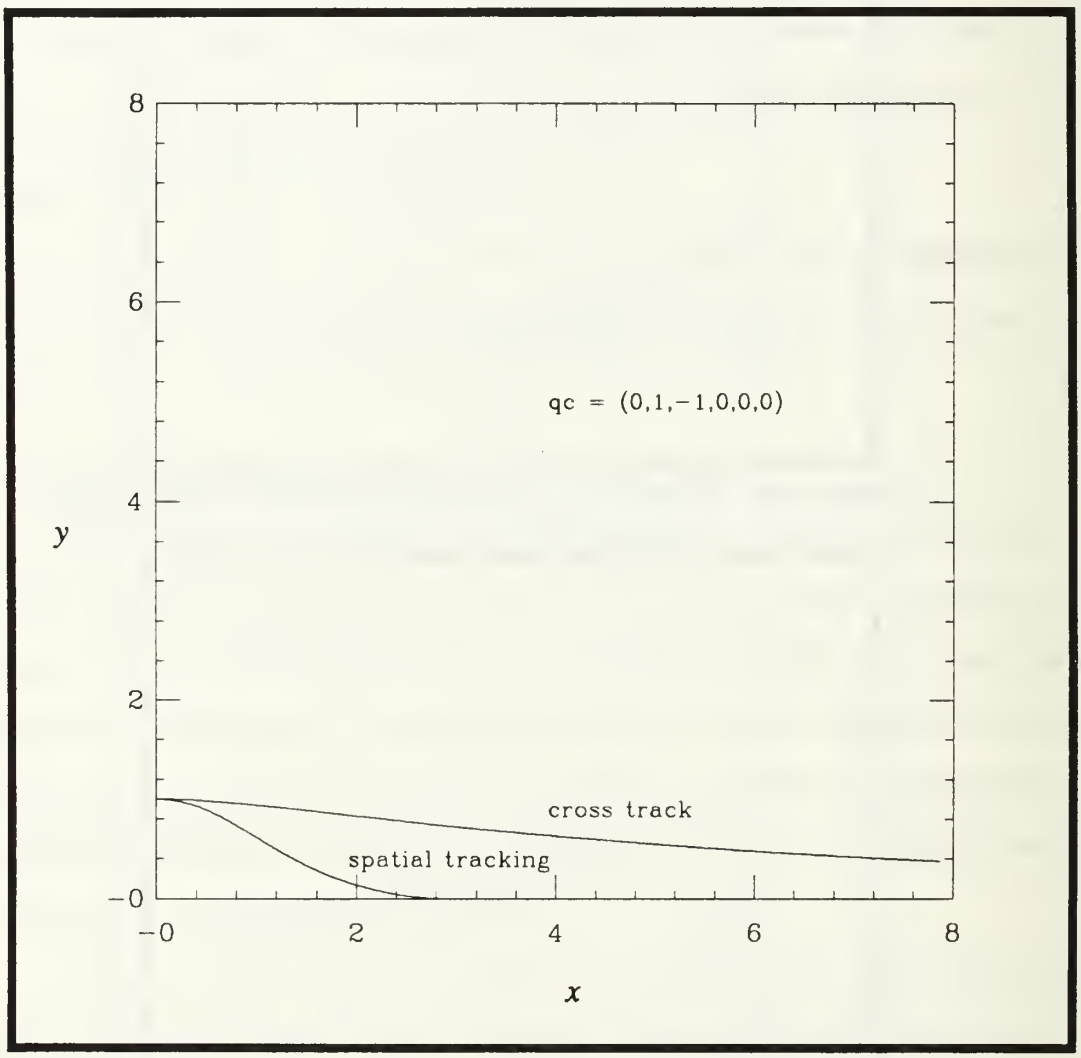

Figure 5-16 **Comparison of Cross Track and Spatial Tracking**

2. Spatial Tracking

By comparing the two methods shown on Figure 5-16, the advantage of spatial tracking should be apparent. This guidance method allows a vehicle to return to the

references path relatively quickly when compared with the convergence of cross track guidance. Also, as previously noted, the expansion of this method to three dimensional applications is simply.

The major disadvantage of spatial tracking is the inability to return to a specific point on the reference path. Since the reference path does not use the concept of a posture, arriving at a desired point is not feasible. This could be a problem in space critical domains.

VI. RESULTS OF SIMULATION

The theory discussed in Chapter V has until now been applied only to a point mass. Since a point mass contains no dynamic characteristics, the test runs conducted in Chapter V were helpful in presenting the spatial tracking theory. However, we are more interested in the presentation of this theory when applied to AUV.

A. SPATIAL TRACKING IN THE SIMULATOR ENVIRONMENT

Prior to the utilization of the spatial tracking algorithm into the simulator environment, two specific modifications to the basic algorithm are required. First, the AUV has certain limitations on maximum allowable curvature and also on maximum rate of curvature change. Second, the vehicle must have the ability to switch reference paths.

1. Limiters

When the spatial tracking theory is applied to an actual vehicle such as the NPS AUV, certain limitations to the commanded curvatures must be introduced. Without these limitations, the guidance system would produce parameters that the vehicle could not obtain. Figure 6-1 is a comparison of a point mass run with and without the two limiters discussed below.

a. Curvature Limitation

As discussed in the section on curvature calculation, the computed curvature value depends upon the position and orientation of the vehicle with respect to the reference path. These commanded values may be impossible for a vehicle to obtain. For example, at maximum fin deflection, the NPS AUV is capable of turning for 360 degrees with a resulting radius of approximately ten feet. Using equation 5-30, the maximum commanded

κ value is 0.1 radians. This means that any commanded value in excess of 0.1 radians in the positive direction or less than -0.1 radians in the negative direction must be reduced to

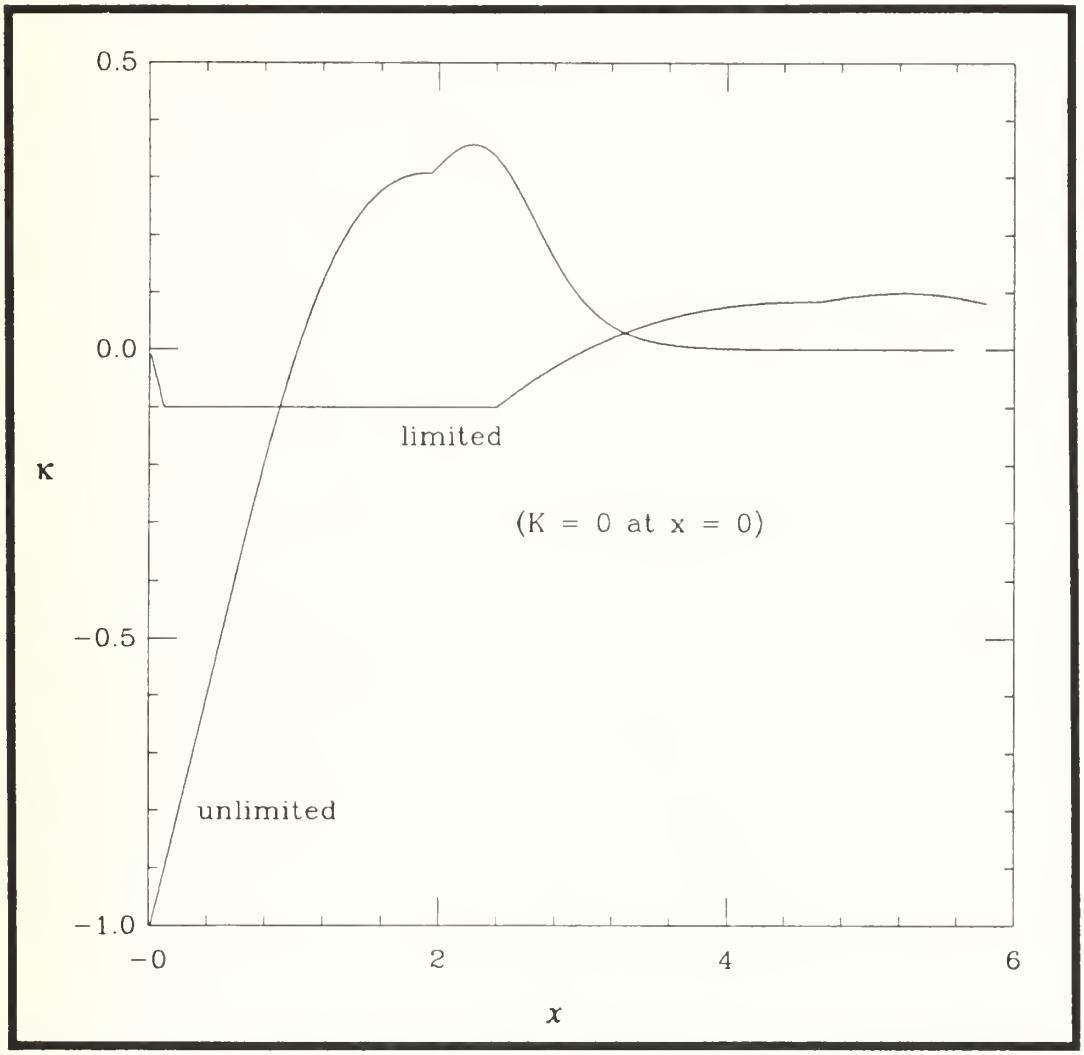

Figure 6-1 Correction to a Reference Path with and without Limiters

this maximum limit. This is accomplished by filtering the curvature calculations in the control loop prior to updating the vehicle's posture at each iteration. See Figure 6-1 for a graphical comparison of point mass runs with and without maximum curvature limiters.

b. Curvature Rate Limitation

With the total range of curvature values limited to a 0.2 radian interval, the rate of change within this interval must also be limited. Since the control loop of the AUV is designed for a ten Hertz cycle, the rate of change of curvature must be constrained. Actual pool tests show that the vehicle's control surfaces may complete a full traversal from full positive angle to full negative angle in one second. Therefore, in 0.1 seconds, the curvature may change 0.02 radians. This limitation is applied to the result of the limited curvature discussed above. The results of this limiter are apparent on Figure 6-1. Notice the gradual change in curvature with the limiter from 0.0 to 0.1 radians as compared with the jump in curvature without the limiter from 0.0 to 1.0 radians.

2. Switching Reference Paths

In Chapter III, the four types of reference paths were introduced. In the point mass examples of Chapter V, the test runs were conducted using a single reference path. In the simulator environment, we are interested in travelling along multiple paths. By combining the four path representations, it is possible to define any path. However, a mechanism must be constructed that allows the switching of the reference from one path to another.

a. Continuous Reference Paths

A continuous reference path is made up of a linked series of waypoints. Each waypoint is connected by a directed ray. The ray must take on the form of one of the four types of reference path representations defined in Chapter III. A waypoint is located wherever two rays cross. This waypoint is also the point that is used to define the ray that is crossed. Figure 3-3 is an example of a continuous reference path. Once a continuous path is defined from a series of waypoints, the mechanism for changing reference from one path to another must be established. There are two possible implementations.

(1) Turning Before Achieving Waypoint. When a turn prior to the waypoint is desired, the mechanism for switching reference paths is when the forrunner point crosses the new reference path. When the forrunner point (A on Figure 5-6) crosses the new reference path, the definition point (q_0) of this new path is used by the guidance system to compute values of curvature. This has the effect of causing the vehicle to turn towards the new reference path a distance of s_0 prior to the waypoint. See Figure 6-2 for a graphical description of this case.

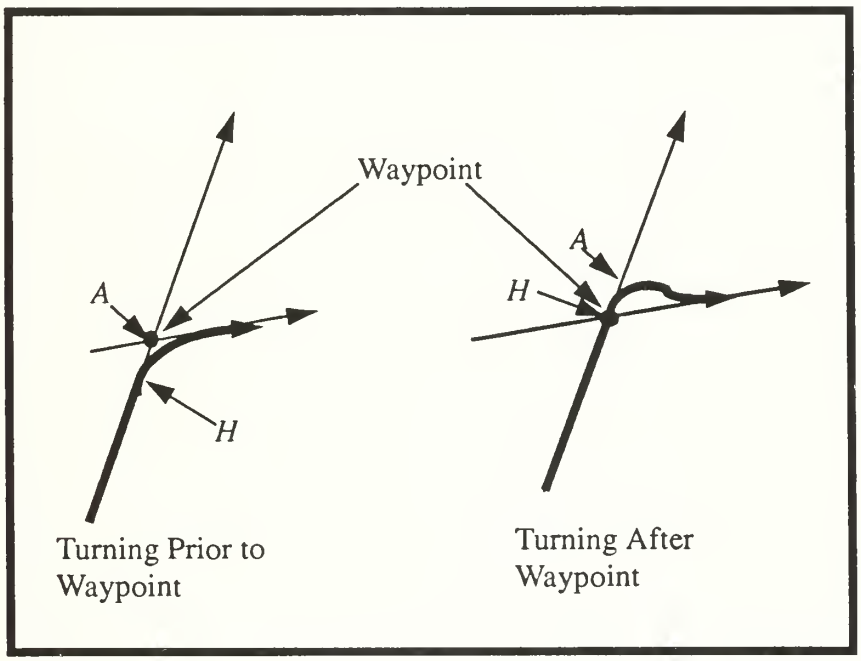

Figure 6-2 **Reference Path Switching**

(2) Turning After Achieving Waypoint. When it is desirable for the vehicle to reach the waypoint prior to turning towards the new path, the mechanism for switching reference paths is the image point (H on Figure 5-6). When the image point reaches the waypoint, the new reference path is used by the guidance system to compute values of curvature. This has the effect of producing an overshoot of the waypoint as may clearly be

seen in Figure 6-2. Also, when this case is used, symmetry is upset when reversing course. See Figure 6-3 for a graphical description of this non-symmetric nature.

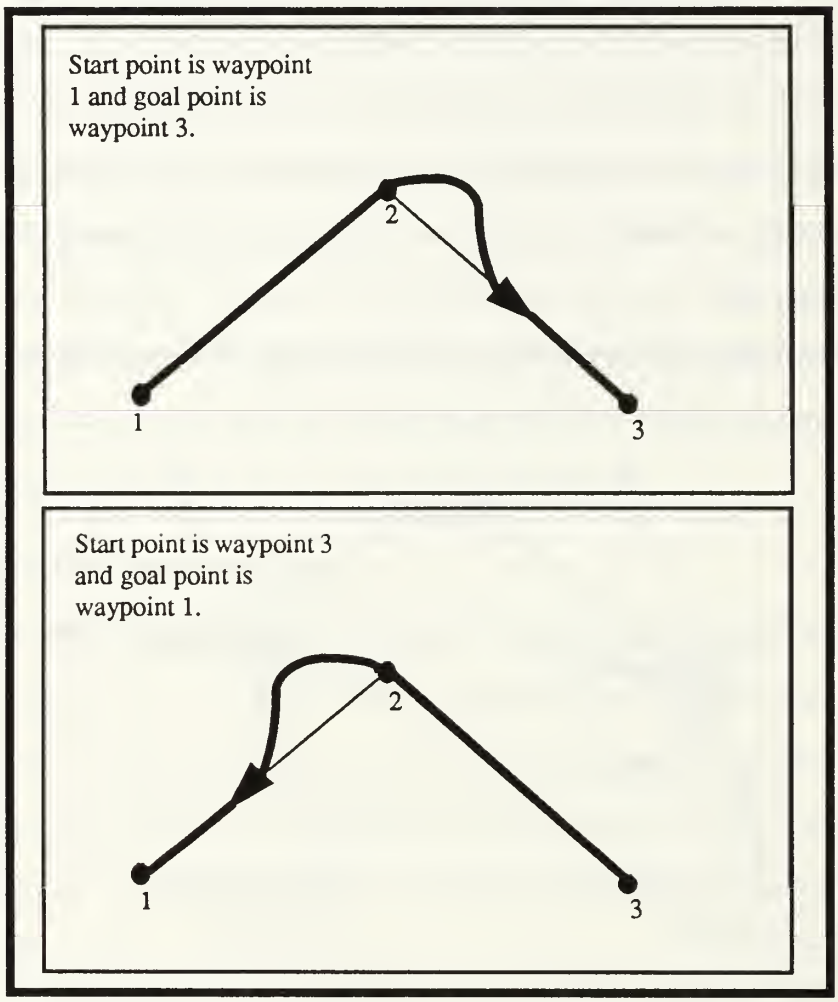

Figure 6-3 Non-Symmetric Nature of Turning After Waypoint

b. Discontinuous Reference Paths

A second type of reference path is a discontinuous reference path. If the discontinuity is the result of a ending half-line (see Figure 3-2), the reference path is switched as the image point arrives at this ending waypoint. However, once we have determined that a switch is required, how do we specify the next reference path? In the continuous case, the next reference path is the next item in the data structure that holds the

waypoints, but because of the discontinuity, this method is unacceptable. In the testbed vehicle, the solution to this problem will be handled outside of the guidance module. Referring to Figure 4-2, the PLAN/REPLAN MISSION module will specify the next reference path. Since this module does not exist in the simulator version, the next reference path must be a user specified input. Once the new reference path is determined, the normal calculation of curvature may take place.

B. RESULTS OF POOL TESTS

Two types of test runs were conducted in the pool environment. These test runs are discussed below.

1. Kinematic Test Results

Prior to implementing the spatial tracking guidance scheme on the simulator with vehicle dynamics, we believe it beneficial to present the results without dynamic constraints. The kinematic path is the ideal path that the vehicle should follow and it is important that the guidance system be independently verified for accuracy prior to interfacing it with other vehicle components.

To achieve a kinematic result, the output of the guidance scheme is feed directly into the H-matrix. Referring to Figure 3-1, this is accomplished by bypassing the autopilot. Figure 6-4 is the ideal kinematic path using waypoints to construct a box pattern within the NPS swimming pool. Waypoints are marked by the letter *W* and a reference line has been added to show the desired path. The resulting path is relatively smooth with very little oscillation. Referring to Figure 6-4, the following parameters were used to produce these results:

* The origin (0, 0, 0) is located at the near left hand corner of the pool floor.
* The positive x- axis is the lower side pool wall.
* The positive y- axis is the left side pool wall.

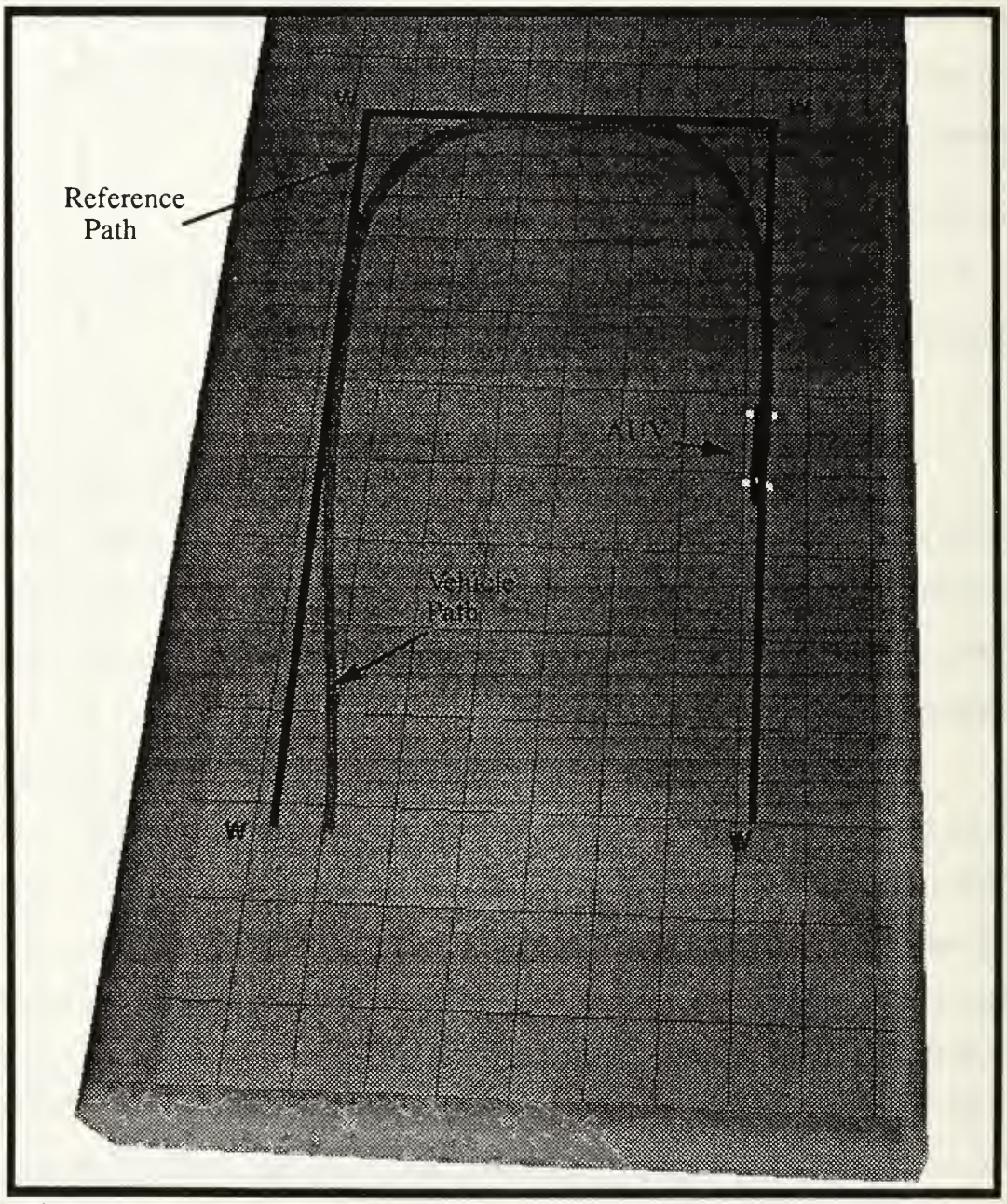

Figure 6-4 **Kinematic Pool Test**

* The positive z- axis is the near left corner measured from the pool floor.

* The waypoints are linked in the order (150, 200, 5), (150, 800, 5), (600, 800, 5), (600, 200, 5).

* $s_0 = 150$.

* $q_c = (200, 200, 5, 0, 0, \frac{\pi}{2})$ (see equation 3-1).

2. Dynamic Test Results

The kinematic test above is very beneficial in verifying the accuracy of the guidance system. However, once this accuracy has been determined dynamic testing of the guidance scheme is required. Figure 6-5 is a test run using the same initial parameters as set forth in the kinematic test of Figure 6-4.

Due to the interface between the components of the simulator, the task of obtaining favorable results was more difficult in the dynamic case than in the kinematic case. In the kinematic case, once an appropriate value for s_0 is found, the test will result in a smooth convergence towards the reference path. However, in the dynamic case, the gains of the autopilot also must be experimented with to achieve a smooth convergence. As evident in Figure 6-5, there is still a slight tendency for the travelled path to contain oscillations. The oscillations may be eliminated by adjusting the gains, but this results in a reduced deflection angle on the rudders and planes of the vehicle. Although we have not included a formal proof, we believe that the current autopilot settings are optimal.

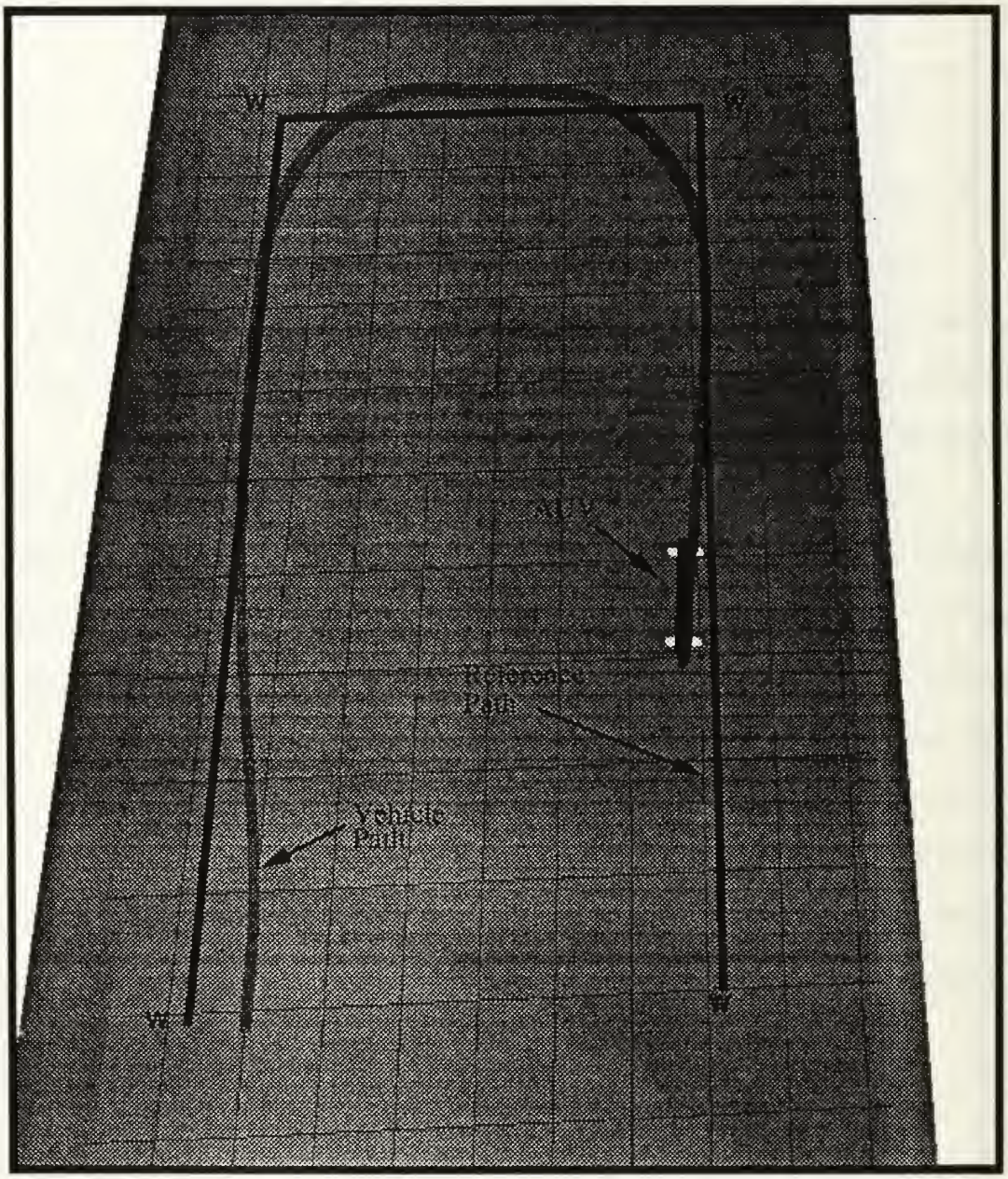

Figure 6-5 **Dynamic Pool Test**

VII. CONCLUSION

The AUV research project at NPS is a continuing, evolutionary process. This work has been just one contribution to the field of AUV research. This chapter is presented here to encourage further research in the field and to point out things that we believe to be essential for follow on research.

A. LESSONS LEARNED

The AUV project at NPS is an extremely complex undertaking. The size of this project makes it virtually impossible for one person to comprehend the full scope of the AUV software and hardware. It was very important from the start to follow proven software development techniques. With the use of a data flow diagram, it was possible for the project to be broken down into individual modules. Members of the research group could then work on an individual module without duplication of effort. The data flow diagrams gave everyone involved a clear understanding of what each module required to function and what each module was responsible for producing. Without this software tool, utilization of time and resources would have been poorly spent.

Accompanying the data flow diagrams for the simulator and the testbed vehicle was a data dictionary. Just as the data flow diagrams were useful in the organization of individual modules, the data dictionary was essential for the description of data. The data dictionary listed individual elements of data structures with their data type. Without it, the individual data flows between modules would be ambiguous. The data dictionary ensured each person working on the project knew what information was to be passed between modules.

B. FUTURE WORK

The NPS AUV is a testbed vehicle. Research in all aspects of the AUV project will continue in the near future. In relation to this work, continued research would be desirable in the following areas.

1. Expanded Use of Simulator Environments

As stated in Chapter III, three different environment models are available for use in the simulator. However, the NPS swimming pool was the only model used within this work. The major reason for this was that this orthogonal environment of the pool was the only one suitable for the linear feature extraction algorithm developed by Floyd [FLOYD 91b]. In parallel with the continued work in linear feature extraction, the utilization of the other environments should be attempted.

2. Expanded Use of Reference Paths

As stated in Chapter III, there are four different representations of a reference path (see Figure 3-2). Within the scope of this work, only the continuous line and half line paths were used in tests. The reasoning behind this was that the path switching mechanisms discussed in Chapter VI have not been developed. Also, the convergence to a curved surface is more complex than the convergence to a line and the necessary modifications to the algorithm must be explored.

APPENDIX A: COMPUTER CODE FOR CROSS TRACK GUIDANCE

```
/*****************************************************************
* control_3d.h
*
* Header file for control_3d.c
*****************************************************************/

#define DELTA_T          0.1
#define LIN_VELOCITY      .50
#define K_x              1.0
#define K1               0.03250
#define K2               5.07
#define K3                .75
#define PI               3.14159265
#define DPI              6.28318530
#define NUM_LOOPS        1500

double delta_theta;

typedef struct ref_posture {
double x;
double y;
double z;
double phi;
double theta;
double psi;
} REF_POS;

typedef struct {
double ref_linear_vel;
double ref_rotate_phi;
double ref_rotate_theta;
double ref_rotate_psi;
} REF_VEL;

typedef struct {
double x;
double y;
double z;
```

```
double phi;
double theta;
double psi;
} CUR_POS;

typedef struct {
double cur_linear_vel;
double cur_rotate_phi;
double cur_rotate_theta;
double cur_rotate_psi;
} CUR_VEL;

typedef struct {
double target_linear_vel;
double target_rotate_phi;
double target_rotate_theta;
double target_rotate_psi;
} VELS;

REF_POS *ref_ptr = NULL;
CUR_POS *cur_ptr = NULL;
VELS *vels_ptr = NULL;
REF_VEL *rvel_ptr = NULL;
CUR_VEL *cvel_ptr = NULL;

CUR_POS *initialize_cur_posture();
REF_POS *initialize_ref_posture();
REF_VEL *initialize_ref_velocities();
VELS *target_velocities();
CUR_VEL *perfect_velocities();
CUR_POS *update_cur_posture();
REF_POS *update_ref_posture();
```

```
/*****************************************************************
* control_3d.c
*
* This is the "C" version of the program that computes linear and
* rotational velocities using the cross track guidance algorithm.
*****************************************************************/

#include <stdio.h>
#include <math.h>
#include "control_3d.h"

main()
{
 int row_index;
 int count = 0;
 double norm();
 FILE *xy_out, *xz_out;

 ref_ptr = initialize_ref_posture();
 rvel_ptr = initialize_ref_velocities();
 cur_ptr = initialize_cur_posture();

 xy_out = fopen("xy3c", "w");
 xz_out = fopen("xz3c", "w");

 for (row_index = 0; row_index < NUM_LOOPS; row_index++) {
 vels_ptr = target_velocities(ref_ptr, cur_ptr, rvel_ptr);
cvel_ptr = perfect_velocities(vels_ptr);
cur_ptr = update_cur_posture(cur_ptr, cvel_ptr);
 ref_ptr = update_ref_posture(ref_ptr);
}

 fprintf(xy_out, "%lf ", cur_ptr->x);
 fprintf(xy_out, " %lf\n", cur_ptr->y);
 fprintf(xz_out, "%lf ", cur_ptr->x);
 fprintf(xz_out, " %lf\n", cur_ptr->z);
 }
 close("xy3c");
 close("xz3c");
}
```

```
CUR_POS *initialize_cur_posture()
{
CUR_POS *cur_ptr;
double deg_psi;
double deg_theta;
double deg_phi;

cur_ptr = (CUR_POS *)malloc(sizeof(CUR_POS));
printf("Enter current posture values.\n");
printf("Enter x-coordinate. Use decimal value.\n");
scanf("%lf", &cur_ptr->x);
printf("\n");
printf("Enter y-coordinate. Use decimal value.\n");
scanf("%lf", &cur_ptr->y);
printf("\n");
printf("Enter z-coordinate. Use decimal value.\n");
scanf("%lf", &cur_ptr->z);
printf("\n");
printf("Enter pitch angle. Use decimal value.\n");
scanf("%lf", &deg_phi);
printf("\n");
printf("Enter roll angle. Use decimal value.\n");
scanf("%lf", &deg_theta);
printf("\n");
printf("Enter yaw angle. Use decimal value.\n");
scanf("%lf", &deg_psi);
printf("\n");
cur_ptr->psi= (deg_psi * 0.01745);
cur_ptr->phi = (deg_phi * 0.01745);
cur_ptr->theta = (deg_theta * 0.01745);
return(cur_ptr);
}
```

```
REF_POS *initialize_ref_posture()
{
REF_POS *ref_ptr;

ref_ptr = (REF_POS *)malloc(sizeof(REF_POS));
ref_ptr->x = 0.0;
ref_ptr->y = 0.0;
ref_ptr->z = 0.0;
ref_ptr->phi = 0.0;
ref_ptr->theta = 0.0;
ref_ptr->psi= 0.0;
return(ref_ptr);
}

REF_VEL *initialize_ref_velocities()
{
REF_VEL *rvel_ptr;

rvel_ptr = (REF_VEL *)malloc(sizeof(REF_VEL));
rvel_ptr->ref_linear_vel = LIN_VELOCITY;
rvel_ptr->ref_rotate_phi = 0.0;
rvel_ptr->ref_rotate_theta = 0.0;
rvel_ptr->ref_rotate_psi = 0.0;
return(rvel_ptr);
}


VELS *target_velocities(ref_ptr, cur_ptr, rvel_ptr)
REF_VEL *rvel_ptr;
REF_POS *ref_ptr;
CUR_POS *cur_ptr;
{
VELS *vels_ptr;
double error_x, error_y, error_z;
double error_phi, error_theta, error_psi;

vels_ptr = (VELS *)malloc(sizeof(VELS));
error_x = (cos(cur_ptr->psi) *  cos(cur_ptr->theta) *
            (ref_ptr->x - cur_ptr->x)) +
          (sin(cur_ptr->psi) *  cos(cur_ptr->theta) *
            (ref_ptr->y - cur_ptr->y)) -
          (sin(cur_ptr->theta) *  (ref_ptr->z - cur_ptr->z));
```

```
error_y = (((cos(cur_ptr->psi) *  sin(cur_ptr->theta) *
              sin(cur_ptr->phi)) -
             (sin(cur_ptr->psi) * cos(cur_ptr->phi))) *
            (ref_ptr->x - cur_ptr->x)) +
           (((sin(cur_ptr->psi) * sin(cur_ptr->theta) *
              sin(cur_ptr->phi)) +
             (cos(cur_ptr->psi) *  cos(cur_ptr->phi))) *
            (ref_ptr->y - cur_ptr->y)) +
           ((cos(cur_ptr->theta) *  sin(cur_ptr->phi)) *
            (ref_ptr->z - cur_ptr->z));

error_z = (((cos(cur_ptr->psi) *  sin(cur_ptr->theta) *
              cos(cur_ptr->phi)) +
             (sin(cur_ptr->psi) * sin(cur_ptr->phi))) *
            (ref_ptr->x - cur_ptr->x)) +
           (((sin(cur_ptr->psi) *  sin(cur_ptr->theta) *
              cos(cur_ptr->phi)) -
             (cos(cur_ptr->psi) *  sin(cur_ptr->phi))) *
            (ref_ptr->y - cur_ptr->y)) +
           ((cos(cur_ptr->theta) * cos(cur_ptr->phi)) *
            (ref_ptr->z - cur_ptr->z));
error_phi = norm(ref_ptr->phi - cur_ptr->phi);
error_theta = norm(ref_ptr->theta - cur_ptr->theta);
error_psi = norm(ref_ptr->psi - cur_ptr->psi);

/*this equations gives target linear velocity*/
 vels_ptr->target_linear_vel = rvel_ptr->ref_linear_vel +
       rvel_ptr->ref_linear_vel * (cos(error_psi) * cos(error_theta) - 1) +
       (K_x * error_x);

/*this equation gives target angular velocity for phi*/
 vels_ptr->target_rotate_phi =
   rvel_ptr->ref_rotate_phi +

 (rvel_ptr->ref_rotate_theta *
 ((sin(ref_ptr->phi) * tan(ref_ptr->theta)) -
 (sin(cur_ptr->phi) * tan(cur_ptr->theta)))) +

 (rvel_ptr->ref_rotate_psi *
 ((cos(ref_ptr->phi) * tan(ref_ptr->theta)) -
 (cos(cur_ptr->phi) * tan(cur_ptr->theta)))) +

 (K1 * sin(error_phi)) +
```

```
( sin(cur_ptr->theta) *
(((error_y * rvel_ptr->ref_linear_vel * cos(error_theta)) /
K3) +

(rvel_ptr->ref_rotate_theta *
((sin(ref_ptr->phi) * tan(ref_ptr->theta)) -
(sin(cur_ptr->phi) * tan(cur_ptr->theta)))) +

(rvel_ptr->ref_rotate_psi *
((cos(ref_ptr->phi) * tan(ref_ptr->theta)) -
(cos(cur_ptr->phi) * tan(cur_ptr->theta)))) +

(K3 * sin(error_psi))));

/*this equation gives target angular velocity for theta*/
 vels_ptr->target_rotate_theta =
rvel_ptr->ref_rotate_theta +

(cos(cur_ptr->phi) * (((- error_z * rvel_ptr->ref_linear_vel) /  K2) +

(rvel_ptr->ref_rotate_theta *
(cos(ref_ptr->phi) - cos(cur_ptr->phi))) +

(rvel_ptr->ref_rotate_psi *
(-sin(ref_ptr->phi) + sin(cur_ptr->phi))) +

(K2 * sin(error_theta)))) +

((sin(cur_ptr->phi) * cos(cur_ptr->theta)) *
(((error_y * rvel_ptr->ref_linear_vel * cos(error_theta)) /  K3) +

(rvel_ptr->ref_rotate_theta *
((sin(ref_ptr->phi) * 1/cos(ref_ptr->theta)) -
(sin(cur_ptr->phi) * 1/cos(cur_ptr->theta)))) +

 (rvel_ptr->ref_rotate_psi *
((cos(ref_ptr->phi) * 1/cos(ref_ptr->theta)) -
(cos(cur_ptr->phi) * 1/cos(cur_ptr->theta)))) +

 (K3 * sin(error_psi))));
```

```
/*this equation gives target angualr velocity for psi*/
 vels_ptr->target_rotate_psi =
 rvel_ptr->ref_rotate_psi +

 (- sin(cur_ptr->phi) *
 (((- error_z * rvel_ptr->ref_linear_vel) /  K2) +

 (rvel_ptr->ref_rotate_theta *
 (cos(ref_ptr->phi) - cos(cur_ptr->phi))) +

 (rvel_ptr->ref_rotate_psi *
 (- sin(ref_ptr->phi) + sin(cur_ptr->phi))) +

 (K2 * sin(error_theta)))) +

 ((cos(cur_ptr->phi) * cos(cur_ptr->theta)) *
 (((error_y * rvel_ptr->ref_linear_vel * cos(error_theta)) /  K2) +

 (rvel_ptr->ref_rotate_theta *
 ((sin(ref_ptr->phi) * 1/cos(ref_ptr->theta)) -
 (sin(cur_ptr->phi) * 1/cos(cur_ptr->theta)))) +

 (rvel_ptr->ref_rotate_psi *
 ((cos(ref_ptr->phi) * 1/cos(ref_ptr->theta)) -
 (cos(cur_ptr->phi) * 1/cos(cur_ptr->theta)))) +

 (K3 * sin(error_psi))));

 return(vels_ptr);
 }

 CUR_POS *update_cur_posture(cur_ptr, cvel_ptr)
 CUR_POS *cur_ptr;
 CUR_VEL *cvel_ptr;
 {
 double delta_s;
 double delta_phi, phi1;
 double delta_theta, theta1;
 double delta_psi, psi1;

 delta_phi = DELTA_T * cvel_ptr->cur_rotate_phi;
 delta_theta = DELTA_T * cvel_ptr->cur_rotate_theta;
 delta_psi = DELTA_T * cvel_ptr->cur_rotate_psi;
```

```
phi1 = cur_ptr->phi + (delta_phi / 2);
theta1 = cur_ptr->theta = (delta_theta / 2);
psi1 = cur_ptr->psi + (delta_psi / 2);

delta_s = DELTA_T * cvel_ptr->cur_linear_vel;

cur_ptr->x = cur_ptr->x + (delta_s * cos(psi1) * cos(phi1));
cur_ptr->y = cur_ptr->y + (delta_s * sin(psi1) * cos(phi1));
cur_ptr->z = cur_ptr->z + (delta_s * sin(phi1));

cur_ptr->phi = cur_ptr->phi + delta_phi;
cur_ptr->theta = cur_ptr->theta + delta_theta;
cur_ptr->psi = cur_ptr->psi + delta_psi;
return(cur_ptr);
}


CUR_VEL *perfect_velocities(vels_ptr)
VELS *vels_ptr;
{
cvel_ptr = (CUR_VEL *)malloc(sizeof(CUR_VEL));
cvel_ptr->cur_linear_vel = vels_ptr->target_linear_vel;
cvel_ptr->cur_rotate_phi = vels_ptr->target_rotate_phi;
cvel_ptr->cur_rotate_theta = vels_ptr->target_rotate_theta;
cvel_ptr->cur_rotate_psi = vels_ptr->target_rotate_psi;
return(cvel_ptr);
}

REF_POS *update_ref_posture(ref_ptr)
REF_POS *ref_ptr;
{
double delta_s;

delta_s = DELTA_T * rvel_ptr->ref_linear_vel;
ref_ptr->x = ref_ptr->x + (delta_s * cos(ref_ptr->psi) * cos(ref_ptr->phi));
ref_ptr->y = ref_ptr->y + (delta_s * sin(ref_ptr->psi) * cos(ref_ptr->phi));
ref_ptr->z = ref_ptr->z + (delta_s * sin(ref_ptr->phi));
ref_ptr->phi = ref_ptr->phi + (DELTA_T * rvel_ptr->ref_rotate_phi);
ref_ptr->theta = ref_ptr->theta + (DELTA_T * rvel_ptr->ref_rotate_theta);
ref_ptr->psi = ref_ptr->psi + (DELTA_T * rvel_ptr->ref_rotate_psi);
return(ref_ptr);
}
```

```
double norm(a)
double a;
{
while ((a > PI) || (a <= -PI)) {
if (a > PI)
a = a - DPI;
else
a = a + DPI;
}
return(a);
}
```

APPENDIX B: COMPUTER CODE FOR SPATIAL TRACKING

```
/*****************************************************************
sim2.h

This is the header file for spat_track.c.
Header file for the AUV Dynamic Simulator based on AUV Dataflow
  Diagram v1.1

*****************************************************************/

#include <stdio.h>
#include <math.h>
#include <time.h>
#define PI                  3.14159265358979323846
#define DPI                 6.28318530717958647692
#define DELTA_T             0.1
#define VELOCITY            2.0
#define s0                  4.0
#define CURVE_MAX           0.1
#define CURVE_DOT_MAX       0.01
#define D_ERROR             0.001



typedef struct {
   double   x,
            y,
            z,
            phi,
            theta,
            psi;
}POINT_3D, POSTURE;

typedef struct{
    POINT_3D     Position;
    int          mode;
}POSTURE_3D;

POSTURE_3D  Ref_posture, Current_posture;
```

```
typedef struct{
    double   fin_deflection[8];
     int         t_main_rpm,
l                t_main_rpm,
               fwd_hor_rpm,
               aft_hor_rpm,
               fwd_ver_rpm,
              aft_ver_rpm;
}CONTROLS;

typedef struct {
 double  x;
 double  y;
 double  z;
 double  phi;
 double  theta;
 double  psi;
 double  kappa;
 double  gamma;
 int         path_gen;

} CONFIG;

CONFIG*config_ptr;

typedef struct path {
 double  x;
 double  y;
 double  z;
 double  na;
 double  phi;
 double  psi;
 double  kappa1;
 double  gamma1;
 int       node_num;
 struct   path *prev;
 struct   path *next;
} REF_PATH;

REF_PATH *top = NULL,  *before = NULL, *now;
FILE *in_file;
```

```
double  limited_kappa, limited_gamma;

REF_PATH      *guidance();
void            update_configuration();
REF_PATH      *compute_kappa();
REF_PATH        *compute_gamma();
double            limiter_kappa();
double            limiter_gamma();
double            norm();
REF_PATH        *create_ref_path();
REF_PATH        *check_for_cross();
double           determine_xy_orientation();
double           determine_xz_orientation();

/*********************************************************************
spat_track.c

This is the spatial tracking guidance code that is installed in the AUV simulator
*********************************************************************/
#include "auv.h"
#include "sim2.h"
#include "init_parameters.h"
REF_PATH *guidance(current_posture,config_ptr, auv,now)
POSTURE_3D *current_posture;
CONFIG *config_ptr;
Sub_ptr *auv;
REF_PATH *now;
{
 int index, limit;
 static double prev_kappa, prev_gamma;

 if((current_posture->Position.psi > PI) && (auv->constraint.box))   {
    current_posture->Position.psi = -(DPI - current_posture->Position.psi);
    config_ptr->psi = current_posture->Position.psi;
    config_ptr->x = current_posture->Position.x;
    config_ptr->y = current_posture->Position.y;
    config_ptr->z = current_posture->Position.z;
   }
  else  {
   if(config_ptr->psi > PI)  {
     config_ptr->psi = -(DPI - config_ptr->psi);

    }
```

```
  }
  if((current_posture->Position.phi > PI) && (auv->constraint.box))  {
    current_posture->Position.phi = -(DPI - current_posture->Position.phi);
    config_ptr->phi = current_posture->Position.phi;
   }
  else {
   if(config_ptr->phi > PI)  {
     config_ptr->phi = -(DPI - config_ptr->phi);
    }
   }
   if(!config_ptr->path_gen)  {
    in_file = fopen("lp.d", "r");
    top = create_ref_path(top, now, in_file);
    config_ptr->path_gen = 1;
    now = top;
    config_ptr->x = -INIT_POOL_POSZ;
    config_ptr->y = INIT_POOL_POSX;
    config_ptr->z = INIT_AUV_POOL_DEPTH;
   }
  now = compute_kappa(config_ptr, now);
  now = compute_gamma(config_ptr, now);
   limited_kappa = limiter_kappa(now, prev_kappa);
  limited_gamma = limiter_gamma(now, prev_gamma);
  prev_kappa = limited_kappa;
  prev_gamma = limited_gamma;
  update_configuration(config_ptr, limited_kappa, limited_gamma, auv);
  return(now);
}


REF_PATH *create_ref_path(top, now, in_file)
REF_PATH *top, *now;
FILE *in_file;
{
  double x, y, z, na;
  double phi, psi;
  int count, node;

  before = NULL;
  fscanf(in_file, "%d", &node);
  for (count = 1; count <= node; count++)  {
   fscanf(in_file, "%lf %lf %lf %lf",
        &x, &y, &z, &na);
```

```
  if (count == 1)     {
    top = (REF_PATH *)malloc(sizeof (REF_PATH));
    before = top;
    now = top;
   }
  else   {
    now->next = (REF_PATH *)malloc(sizeof(REF_PATH));
    now = now->next;
   }
  now->x = x;
  now->y = y;
  now->z = z;
  now->na = na;

  now->psi = determine_xy_orientation(now, before);
  now->phi = determine_xz_orientation(now, before);
  now->prev = before;
  before = now;
  now->node_num = node;
  now->next = NULL;
 }
 top->prev = now;
 top->node_num = count;
 now->next = top;
 return(top);
}


double determine_xy_orientation(now, before)
REF_PATH *now, *before;
{
 before->psi = atan2((now->y - before->y), (now->x - before->x));
 return(before->psi);
}


double determine_xz_orientation(now, before)
REF_PATH *now, *before;
{
 before->phi = atan2((now->z - before->z), (now->x - before->x));
 return(before->phi);
}
```

```
double limiter_kappa(now, prev_kappa)
REF_PATH *now;
double prev_kappa;
{
 if (now->kappa1 > CURVE_MAX) {
  now->kappa1 = CURVE_MAX;
 }
 else if (now->kappa1 < - CURVE_MAX) {
  now->kappa1 = - CURVE_MAX;
 }
 if (prev_kappa >= 0.0) {
  if (now->kappa1 >= 0.0) {
   if ((now->kappa1 >= (prev_kappa + CURVE_DOT_MAX)) &&
      (now->kappa1 > prev_kappa)) {
    now->kappa1 = prev_kappa + CURVE_DOT_MAX;
   }
   else if ((now->kappa1 < (prev_kappa - CURVE_DOT_MAX)) &&
         (now->kappa1 < prev_kappa)) {
    now->kappa1 = prev_kappa - CURVE_DOT_MAX;
   }
  }
  else if (now->kappa1 < 0.0) {
   if (now->kappa1 < (prev_kappa - CURVE_DOT_MAX)) {
    now->kappa1 = prev_kappa - CURVE_DOT_MAX;
   }
  }
 }
 else if (prev_kappa < 0.0) {
  if (now->kappa1 >= 0.0) {
   if (now->kappa1 >= (prev_kappa + CURVE_DOT_MAX)) {
    now->kappa1 = prev_kappa + CURVE_DOT_MAX;
   }
  }
  else if (now->kappa1 < 0.0) {
   if ((now->kappa1 >= (prev_kappa + CURVE_DOT_MAX)) &&
      (prev_kappa < now->kappa1)) {
    now->kappa1 = prev_kappa + CURVE_DOT_MAX;
   }
   else if ((now->kappa1 < (prev_kappa - CURVE_DOT_MAX)) &&
         (prev_kappa > now->kappa1)) {
    now->kappa1 = prev_kappa - CURVE_DOT_MAX;
   }
  }
```

```
 }
 return(now->kappa1);
}


double limiter_gamma(now, prev_gamma)
REF_PATH *now;
double prev_gamma;

{

 if (now->gamma1 > CURVE_MAX) {
  now->gamma1 = CURVE_MAX;
 }
 else if (now->gamma1 < - CURVE_MAX) {
  now->gamma1 = - CURVE_MAX;
 }
 if (prev_gamma >= 0.0) {
  if (now->gamma1 >= 0.0) {
   if ((now->gamma1 >= (prev_gamma + CURVE_DOT_MAX)) &&
      (now->gamma1 > prev_gamma)) {
    now->gamma1 = prev_gamma + CURVE_DOT_MAX;
   }
   else if ((now->gamma1 < (prev_gamma - CURVE_DOT_MAX)) &&
         (now->gamma1 < prev_gamma)) {
    now->gamma1 = prev_gamma - CURVE_DOT_MAX;
   }
  }
  else if (now->gamma1 < 0.0) {
   if (now->gamma1 < (prev_gamma - CURVE_DOT_MAX)) {
    now->gamma1 = prev_gamma - CURVE_DOT_MAX;
   }
  }
 }
 else if (prev_gamma < 0.0) {
  if (now->gamma1 >= 0.0) {
   if (now->gamma1 >= (prev_gamma + CURVE_DOT_MAX)) {
    now->gamma1 = prev_gamma + CURVE_DOT_MAX;
   }
  }
  else if (now->gamma1 < 0.0) {
   if ((now->gamma1 >= (prev_gamma + CURVE_DOT_MAX)) &&
      (prev_gamma < now->gamma1)) {
```

```
      now->gamma1 = prev_gamma + CURVE_DOT_MAX;
     }
     else if ((now->gamma1 < (prev_gamma - CURVE_DOT_MAX)) &&
           (prev_gamma > now->gamma1)) {
      now->gamma1 = prev_gamma - CURVE_DOT_MAX;
     }
    }
  }
  return(now->gamma1);
}


REF_PATH *compute_kappa(config_ptr, now)
CONFIG *config_ptr;
REF_PATH *now;
{
  double psi1, alpha, beta, temp_kappa;
  double omega, beta1;
  double d, d0, d1, s1;
  double q0x, q0y, q0psi;
  double Ax, Ay, Hx, Hy;

  typedef struct {
   doublex;
   doubley;
  } REF;

  REF *image, *ahead;
  REF_PATH *before;

  q0x = now->x;
  q0y = now->y;
  q0psi = now->psi;

  d0 = sqrt(fabs(pow(fabs(q0x - config_ptr->x), 2.0) +
         pow(fabs(q0y - config_ptr->y), 2.0)));

  beta1 = atan2((q0y - config_ptr->y), (q0x - config_ptr->x ));
  beta = beta1 - (q0psi - PI/2);

  d1 = d0 * sin(beta);
  image = (REF *)malloc(sizeof(REF));
```

```
image->x = (q0x - d1 * cos(q0psi));
image->y = (q0y - d1 * sin(q0psi));

ahead = (REF *)malloc(sizeof(REF));
ahead->x = (image->x + s0 * cos(q0psi));
ahead->y = (image->y + s0 * sin(q0psi));

alpha = atan2((ahead->y - config_ptr->y),
          (ahead->x - config_ptr->x));

psi1 = (2 * alpha) - config_ptr->psi;
omega = alpha - config_ptr->psi;
d = sqrt(fabs(pow(fabs(ahead->x - config_ptr->x), 2.0) +
       pow(fabs(ahead->y - config_ptr->y), 2.0)));

if (config_ptr->y >= q0y) {
 if (psi1 <= q0psi){
  now->kappa1 =  2 * sin(omega) / d;
 }
 else  {
     now->kappa1 = ((pow((1.0 - cos(config_ptr->psi - q0psi)), 2.0)) *
          s0) /
         ((d0 * cos(beta)) * fabs((d0 * cos(beta)) *
                        sin(config_ptr->psi - q0psi)));
  }
 }
else if (config_ptr->y < q0y) {
    if (psi1 >= q0psi)  {
      now->kappa1 = 2 * sin(omega) / d;
     }
    else {
      now->kappa1 = ((pow((1.0 - cos(config_ptr->psi - q0psi)), 2.0)) *
           s0) /
         ((d0 * cos(beta)) * fabs((d0 * cos(beta)) *
                        sin(config_ptr->psi - q0psi)));
     }
}
temp_kappa = now->kappa1;
Ax = ahead->x;
Ay = ahead->y;
Hx = image->x;
Hy = image->y;
before = now;
```

```
 now = now->next;
 now = check_for_cross(now, before, Ax, Ay, Hx, Hy);
 now->kappa1 = temp_kappa;
 return(now);
}


REF_PATH *compute_gamma(config_ptr, now)
CONFIG *config_ptr;
REF_PATH *now;
{
 double phi1, alpha, beta, temp_gamma;
 double omega, beta1;
 double d, d0, d1;
 double q0x, q0z, q0phi;
 double Ax, Az, Hx, Hz;
 typedef struct {
  doublex;
  doublez;
 } REF;
 REF *image, *ahead;
 REF_PATH *before;

 q0x = now->x;
 q0z = now->z;
 q0phi = now->phi;
 d0 = sqrt(fabs(pow(fabs(q0x - config_ptr->x), 2.0) +
        pow(fabs(q0z - config_ptr->z), 2.0)));
 beta1 = atan2((q0z - config_ptr->z), (q0x - config_ptr->x));
 beta = beta1 - (q0phi - PI/2);
 d1 = d0 * sin(beta);
 image = (REF *)malloc(sizeof(REF));
 image->x = (q0x - d1 * cos(q0phi));
 image->z = (q0z - d1 * sin(q0phi));
 ahead = (REF *)malloc(sizeof(REF));
 ahead->x = (image->x + s0 * cos(q0phi));
 ahead->z = (image->z + s0 * sin(q0phi));
 alpha = atan2((ahead->z - config_ptr->z),
           (ahead->x - config_ptr->x));
 phi1 = (2 * alpha) - config_ptr->phi;
 omega = alpha - config_ptr->phi;
 d = sqrt(fabs(pow(fabs(ahead->x - config_ptr->x), 2.0) +
        pow(fabs(ahead->z - config_ptr->z), 2.0)));
```

```
 if (config_ptr->z >= q0z) {
  if (phi1 <= q0phi) {
   now->gamma1 =  2 * sin(omega) / d;
  }
  else if (phi1 > q0phi) {
   now->gamma1 = ((pow((1.0 - cos(config_ptr->phi - q0phi)), 2.0)) *
            s0) /
           ((d0 * cos(beta)) * fabs((d0 * cos(beta)) *
                        sin(config_ptr->phi - q0phi)));
  }
 }
 else if (config_ptr->z < q0z) {
  if (phi1 >= q0phi) {
   now->gamma1 = 2 * sin(omega) / d;
  }
  else if (phi1 < q0phi) {
   now->gamma1 = ((pow((1 - cos(config_ptr->phi - q0phi)), 2.0)) *
            s0) /
           ((d0 * cos(beta)) * fabs((d0 * cos(beta)) *
                        sin(config_ptr->phi - q0phi)));
  }
 }
 temp_gamma = now->gamma1
 Ax = ahead->x;
 Az = ahead->z;
 Hx = image->x;
 Hz = image->z;
 before = now;
 now = now->next;
 now = check_for_cross(now, before, Ax, Az, Hx, Hz);
 now->gamma1 = temp_gamma;
 return(now);
}
REF_PATH *check_for_cross(now, before, Ax, Ay, Hx, Hy)
REF_PATH *now, *before;
double Ax, Ay, Hx, Hy;
{
 double psi_d, beta;
 double d0, d1, dw;

 typedef struct {
  double x;
```

```
 doubley;
} CROSS;


CROSS *w, *c;

psi_d = now->psi - before->psi;
d0 = sqrt(fabs(pow(fabs(now->x - before->x), 2.0) +
        pow(fabs(now->y - before->y), 2.0)));
beta = atan2((now->y - before->y), (now->x -
          before->x)) - (before->psi);
d1 = d0 * sin(beta);

w = (CROSS *)malloc(sizeof(CROSS));
w->x = (now->x - d1 * cos(before->psi + PI/2));
w->y = (now->y - d1 * sin(before->psi + PI/2));

dw = d1 / tan(psi_d);

c = (CROSS *)malloc(sizeof(CROSS));
c->x = (w->x - dw * cos(before->psi));
c->y = (w->y - dw * sin(before->psi));

if (fabs(fabs(Hx) - fabs(c->x)) < D_ERROR)
  Hx = c->x;
if (fabs(fabs(Hy) - fabs(c->y)) < D_ERROR)
  Hy = c->y;
if (fabs(fabs(Ax) - fabs(c->x)) < D_ERROR)
  Ax = c->x;
if (fabs(fabs(Ay) - fabs(c->y)) < D_ERROR)
  Ay = c->y;

 if (((Hx >= c->x) && (Ax >= c->x)) && ((Hy >= c->y) && (Ay >= c->y)) ||
    (((Hx <= c->x) && (Ax <= c->x)) && ((Hy <= c->y) && (Ay <= c->y)))) {
  return(before);
 }
 else {
  return(now);
 }
}
```

```
void update_configuration(config_ptr, limited_kappa, limited_gamma, auv)
CONFIG *config_ptr;
double limited_kappa;
double limited_gamma;
Sub_ptr *auv;
{
 double delta_s;
 double delta_psi;
 double delta_phi;

 if(auv->constraint.box)
  delta_s = DELTA_T * 24.0;
 else
  delta_s = DELTA_T * auv->dyn.vel[0] * 12.0;
 delta_psi = delta_s * config_ptr->kappa;
 delta_phi = delta_s * config_ptr->gamma;
 config_ptr->x = config_ptr->x + (delta_s * cos(config_ptr->psi +
                                  delta_psi / 2)); *
                          cos(config_ptr->phi));
 config_ptr->y = config_ptr->y + (delta_s * sin(config_ptr->psi +
                                  delta_psi / 2)); *
                          cos(config_ptr->phi));
 config_ptr->z = config_ptr->z + (delta_s * sin(config_ptr->phi));
 config_ptr->kappa = limited_kappa;
 config_ptr->gamma = limited_gamma;
 config_ptr->phi = config_ptr->phi + (delta_s * config_ptr->gamma);
 config_ptr->psi = config_ptr->psi + (delta_s * config_ptr->kappa);
}


double norm(a)
double a;
{
 while ((a > PI) || (a <= -PI)) {
  if (a > PI)
   a = a - DPI;
  else
   a = a + DPI;
 }
 return(a);
}
```

APPENDIX C: NPS AUV DYNASIMUSER'S MANUAL

I. INTRODUCTION

The following document is intended to aid the user of the NPS AUV DYNASIM. This 3D graphic simulator was designed and coded by Tom Jurewicz as his thesis project in December of 1990. Further additions to this simulator were made by Floyd, Magrino, Brutzman, and Caddell throughout 1991. Accompanying this manual is an on-line user's manual that gives the user ready access to information by depressing the right mouse over any menu item that needs explanation.

II. TABLE OF CONTENTS

III. USER'S MANUAL

The following is a detailed list of instructions for each menu item.

A. Velocities

When the left mouse is activated on the velocities option, the Velocity panel will appear in the upper left hand corner of the screen. A velocity for each degree of the six degrees of freedom that the AUV may move is displayed:

1) Surge

2) Sway

3) Heave

4) Roll

5) Pitch

6) Yaw

This panel is utilized to monitor the state of the vehicle while running dynamic tests. Comparisons with data obtained from in-pool testing should reveal whether or not the vehicle is responding appropriately.

The numeric box above each of the six graphs gives accelerations for each of the six degrees of freedom in the AUV. The numeric box below each of the six graphs gives velocities for each of the six degrees of freedom in the AUV. The red line on the chart represents zero acceleration and the black line represents history of the changes in acceleration for the current test.

The box in the lower left hand corner of this panel allows the user to hide this panel.

B. Recorder

When the left mouse is depressed on the Recorder option, the Recorder panel will appear in the bottom left hand corner of the screen. The recorder provides the capability to

record and replay scenarios. The recording is made in the ASCII file "recording" which contains the initial vehicle state followed by times, RPMs and fin deflection whenever a change of RPM or deflection occurred. The panel is hidden by depressing the left mouse button in the box of the lower left hand corner.

1. Off

When this button is active (an X appears in the box to the left), no record is being made of the current simulator mission. If this button is inactive (no X), either the Record or Playback button must be active.

2. Record

When this button is active (an X appears in the box to the left), the current simulator mission is being recorded. Times, RPMs and fin deflections are being recorded in the "recording" file in ASCII text. When a recording is made, it erases the previous tape named "recording."

3. Play

When this button is active (an X appears in the box to the left), the latest simulator mission to be recorded will be played back. Playbacks may occur as many times as desired without erasing the tape. External scripts may be played if they are loaded to the "recording" file.

4. Auxiliary

This feature is still under development.

5. Speed

This option allows the user to adjust playback speed. The speed ranges may be adjusted from zero to five times normal speed.

C. Coefficients

When the left mouse is depressed on the Coefficients option, the Coefficients panel will appear in the bottom left hand corner of the screen. The Coefficients panel provides

the user the capability to modify the Hydrodynamic and Added Mass Coefficients while watching the effects upon the vehicle on the screen. One of seven different actuator panels may be called up by selecting the appropriate button. The base coefficient information is stored in the "coefficients.dir" directory and are accessed by the simulator to provide accurate vehicle motion. The panel has 10 selector buttons on the bottom that have the following functions:

Save File: Allows the user to save a copy of changes that were made to the coefficients. This file is placed in the coefficients.dir directory and overwrites any existing file by that name.

Read File: Allows the user to recall a previously saved file for use by the simulator. When this file is recalled, the base values on the actuators will be replaced with the saved values.

Exit: Hides the actuators. The exit box in the lower left corner of the panel only hides the menu panel.

The remaining seven buttons allow the user to select which actuator panel is to be used.

Added Mass: Allows user to change the mass coefficients of the simulator vehicle. This is the panel that is initially displayed

Surge: When selected, displays Surge Hydrodynamic Coefficients.

Sway: When selected, displays Sway Hydrodynamic Coefficients.

Heave: When selected, displays Heave Hydrodynamic Coefficients.

Roll: When selected, displays Roll Hydrodynamic Coefficients.

Pitch: When selected, displays Pitch Hydrodynamic Coefficients.

Yaw: When selected, displays Yaw Hydrodynamic Coefficients.

D. Frames

When the left mouse is depressed on the Frames option, the Frames panel will appear in the bottom left hand corner of the screen. The Frames panel gives the workstation's performance in two ways. Delta Time is the total time between swapbuffers and the frame rate is the inverse, i.e., total frames per second. The meters are single pen stripcharts.

E. Dynamics

When button is selected, the simulator switches from kinematic input from spaceball/ mouse to dynamic input from H-matrix.

F. Cockpit View

This option gives the user the ability to view the simulation from one of two views. When the Cockpit View button is active (an X appears in the box to the left of the menu choice), the simulation view is from the nose of the vehicle.

When the Cockpit View button is inactive (no X appears in the box to the left of the menu choice), the simulation view is external to the vehicle. The actual view position in this mode is dependent upon the settings of the five dials above the main menu panel:

* Inclination
* Azimuth
* Distance (near)
* Distance (far)
* Twist

The cockpit view is toggled from active to inactive by depressing the left mouse button over the menu box.

G. AUV Center

This option gives the user the ability to position the vehicle in relation to the screen. When the AUV Center button is active (an X appears in the box to the left of the menu choice), the vehicle remains in the center of the viewing screen while the background

terrain moves. This option gives the user the perspective that he is moving with the vehicle as it moves.

When the Cockpit View button is inactive (no X appears in the box to the left of the menu choice), the vehicle moves while the terrain stays motionless. This option gives the user the perspective that he is stationary as the vehicle moves. The AUV Center button is toggled from active to inactive by depressing the left mouse button over the menu box.

H. Sonar

When the left mouse is depressed on the Sonar option, the Sonar panel will appear in the upper right hand corner of the screen. The Sonar panel has several features:

Sonar Selection: The five boxes at the top of the panel labeled one thru five allow the user to select the use of individual sonars. Any combination of sonars may be selected from one to all. At present, the vehicle only utilizes four sonars. The fifth box is added for flexibility.

Speed and Heading: The two dials indicate vehicle speed and heading. Also present above each dial is a numeric value that gives a precise value of the corresponding dial.

Floor and Depth: The Floor and Depth bar graphs give a reading of the depth of the pool floor from the surface and the depth of the vehicle from the surface respectively. The floor bar graph bottoms out at -8 feet. If the actual bottom is less than this value, a solid fill is added from -8 feet to the actual depth. The corresponding numeric values are also displayed above the bar graphs.

Bottom Contour: This gives a two dimensional view of the pool bottom (blue line) and the vehicle depth (red line). This graph is plotted only while the current sonar panel is displayed. All previous graphs are cleared when the user exits the sonar box.

Exit: A left mouse hit to the square in the bottom left hand corner allows the user to exit from the sonar panel.

I. Mission Planner

This panel is still under development.

J. Control Panel

When the left mouse is depressed on the Control Panel option, the Control Panel will appear above the main menu. Each element of this panel is explained individually by depressing the right mouse on the menu option of this panel.

1. Box

When the left mouse is depressed on this box, the user loses manual control and a preprogrammed box pattern is run inside the pool.

2. Snake

When the left mouse is depressed on this box, the user loses manual control and a preprogrammed snake pattern is run inside the pool.

3. Figure 8

When the left mouse is depressed on this box, the user loses manual control and a preprogrammed Figure 8 pattern is run inside the pool.

4. Rudders

When the left mouse is depressed on this bar and the Mouse box is active, the user is able to adjust the rudders on the vehicle to the desired angle. The top bar indicates the left rudders and the bottom bar indicates the right rudders. Unless the SepRuds box is active, any input on the left rudder will cause an opposite reaction on the right rudder. This also applies in reverse. If the SepRuds box is not active (no X), an input to one side results in only a rudder movement on that side. On initial start-up of the simulation, rudders are positioned at 0.0 degrees. The box to the left of each rudder bar gives the actual bar position to the nearest 10th of a degree. The rudders have a deflection range of -40.0 degrees to 40.0 degrees.

There are two ways in which the user may use this bar. The first is a snap position where the user depresses the left mouse button anywhere within the bar and the when the button is released, the viewing angle snaps to this degree position. The second way is to depress the left mouse button over the current position of the bar and drag it to the desired angle.

5. Planes

When the left mouse is depressed on this bar and the Mouse box is active, the user is able to adjust the planes on the vehicle to the desired angle. The left bar indicates the left planes and the right bar indicates the right planes. Unless the SepRuds box is active, any input on the left planes will cause an opposite reaction on the right planes. This also applies in reverse. If the SepRuds box is not active (no X), an input to one side results in only a plane movement on that side. On initial start-up of the simulation, planes are positioned at 0.0 degrees. The box to the bottom of each plane bar gives the actual bar position to the nearest 10th of a degree. The planes have a deflection range of -40.0 degrees to 40.0 degrees.

There are two ways in which the user may use this bar. The first is a snap position where the user depresses the left mouse button anywhere within the bar and the when the button is released, the viewing angle snaps to this degree position. The second way is to depress the left mouse button over the current position of the bar and drag it to the desired angle.

6. Mouse

When this box is active (an X is present in the box), the movement of the rudders and planes bars are controlled by the mouse. If the box is not active (no X), the rudders and planes are controlled by the spaceball.

7. Sep Ruds

When the SepRuds and the Mouse boxes are active (an X is present in both boxes), the user is able to independently control the left and the right rudders of the vehicle. When this box is not active, an input to one rudder gives the opposite corresponding input to the other side.

8. Sep Planes

When the SepPlanes and the Mouse boxes are active (an X is present in both boxes), the user is able to independently control the left and the right planes of the vehicle. When this box is not active, an input to one plane gives the opposite corresponding input to the other side.

9. Sep RPM

When the SepRPM box is active (an X is present in the box), the user is able to independently control each of the six RPM bars on the control panel. When this box is not active, the six RPMs are paired into three groups:

* L.M.RPM and the R.M.RPM (left main and right main).

* F.H.RPM and the R.H.RPM (front hover and rear hover).

* F.V.RPM and the R.V.RPM (front vertical and rear vertical).

An input to one RPM of the pair causes the other RPM in the pair to advance to the same position.

10. Neutral

When the left mouse is depressed on this box, it causes all plane, rudder and RPM settings to return to the equilibrium state of zero.

11. Soft Constants

This box is still under development.

12. Hard Constants

This box is still under development.

13. Auto Depth

When the Auto Depth and the Mouse boxes are active (an X is present in both boxes), the depth of the vehicle will stay at the last value recorded. Any adjustments to the planes bars will have no effect upon vehicle motion.

14. Auto Course

When the Auto Course and the Mouse boxes are active (an X is present in both boxes), the heading of the vehicle will stay at the last value recorded. Any adjustments to the rudders bars will have no effect upon vehicle motion.

15. Auto Speed

When the Auto Speed box is active (an X is present in the box), the speed of the vehicle will stay at the last value recorded. Any adjustments to the RPM bars will have no effect upon vehicle speed.

16. L. M. RPM

When the left mouse is depressed on this bar, the user is able to adjust the RPM of the left propeller. If the SepRPM box is active (an X is present in the box to the left of it), an adjustment to left main RPM may be accomplished without effecting any settings on the right main RPM. If the SepRPM box is not active (no X), any input to the L.M. RPM will cause the same input to be applied to the R.M. RPM. On initial start-up of the simulation, RPMs are positioned at 0.0 degrees. The box to the left of each RPM bar gives the actual bar position to the nearest 10th of a RPM. The RPM bars have a deflection range of -1000.0 RPMs to 1000.0 RPMs.

There are two ways in which the user may use this bar. The first is a snap position where the user depresses the left mouse button anywhere within the bar and the when the button is released, the viewing angle snaps to this degree position. The second way is to

depress the left mouse button over the current position of the bar and drag it to the desired angle.

17. R. M. RPM

When the left mouse is depressed on this bar, the user is able to adjust the RPM of the right propeller. If the SepRPM box is active (an X is present in the box to the left of it), an adjustment to right main RPM may be accomplished without effecting any settings on the left main RPM. If the SepRPM box is not active (no X), any input to the R.M. RPM will cause the same input to be applied to the L.M. RPM. On initial start-up of the simulation, RPMs are positioned at 0.0 degrees. The box to the left of each RPM bar gives the actual bar position to the nearest 10th of a RPM. The RPM bars have a deflection range of -1000.0 RPMs to 1000.0 RPMs.

There are two ways in which the user may use this bar. The first is a snap position where the user depresses the left mouse button anywhere within the bar and the when the button is released, the viewing angle snaps to this degree position. The second way is to depress the left mouse button over the current position of the bar and drag it to the desired angle.

18. F. H. RPM

When the left mouse is depressed on this bar, the user is able to adjust the RPM of the forward hover thruster. If the SepRPM box is active (an X is present in the box to the left of it), an adjustment to forward hover RPM may be accomplished without effecting any settings on the rear hover RPM. If the SepRPM box is not active (no X), any input to the F.H. RPM will cause the same input to be applied to the R.H. RPM. On initial start-up of the simulation, RPMs are positioned at 0.0 degrees. The box to the left of each RPM bar gives the actual bar position to the nearest 10th of a RPM. The RPM bars have a deflection range of -1000.0 RPMs to 1000.0 RPMs.

There are two ways in which the user may use this bar. The first is a snap position where the user depresses the left mouse button anywhere within the bar and the when the button is released, the viewing angle snaps to this degree position. The second way is to depress the left mouse button over the current position of the bar and drag it to the desired angle.

19. R. H. RPM

When the left mouse is depressed on this bar, the user is able to adjust the RPM of the rear hover thruster. If the SepRPM box is active (an X is present in the box to the left of it), an adjustment to rear hover RPM may be accomplished without effecting any settings on the forward hover RPM. If the SepRPM box is not active (no X), any input to the R.H. RPM will cause the same input to be applied to the F.H. RPM. On initial start-up of the simulation, RPMs are positioned at 0.0 degrees. The box to the left of each RPM bar gives the actual bar position to the nearest 10th of a RPM. The RPM bars have a deflection range of -1000.0 RPMs to 1000.0 RPMs.

There are two ways in which the user may use this bar. The first is a snap position where the user depresses the left mouse button anywhere within the bar and the when the button is released, the viewing angle snaps to this degree position. The second way is to depress the left mouse button over the current position of the bar and drag it to the desired angle.

20. F. V. RPM

When the left mouse is depressed on this bar, the user is able to adjust the RPM of the forward vertical thruster. If the SepRPM box is active (an X is present in the box to the left of it), an adjustment to forward vertical RPM may be accomplished without effecting any settings on the rear vertical RPM. If the SepRPM box is not active (no X), any input to the F.V. RPM will cause the same input to be applied to the R.V. RPM. On initial start-up of the simulation, RPMs are positioned at 0.0 degrees. The box to the left of each

RPM bar gives the actual bar position to the nearest 10th of a RPM. The RPM bars have a deflection range of -1000.0 RPMs to 1000.0 RPMs.

There are two ways in which the user may use this bar. The first is a snap position where the user depresses the left mouse button anywhere within the bar and the when the button is released, the viewing angle snaps to this degree position. The second way is to depress the left mouse button over the current position of the bar and drag it to the desired angle.

21. R. V. RPM

When the left mouse is depressed on this bar, the user is able to adjust the RPM of the rear vertical thruster. If the SepRPM box is active (an X is present in the box to the left of it), an adjustment to rear vertical RPM may be accomplished without effecting any settings on the forward vertical RPM. If the SepRPM box is not active (no X), any input to the R.V. RPM will cause the same input to be applied to the F.V. RPM. On initial start-up of the simulation, RPMs are positioned at 0.0 degrees. The box to the left of each RPM bar gives the actual bar position to the nearest 10th of a RPM. The RPM bars have a deflection range of -1000.0 RPMs to 1000.0 RPMs.

There are two ways in which the user may use this bar. The first is a snap position where the user depresses the left mouse button anywhere within the bar and the when the button is released, the viewing angle snaps to this degree position. The second way is to depress the left mouse button over the current position of the bar and drag it to the desired angle.

K. Pool

This option is hard coded on in this version of the simulator. In future versions, the user will be able to select three different terrains:

* The NPS Swimming Pool

* Monterey Bay (200 meter spacing)

* Monterey Harbor (30 feet spacing)

L. Execute Mission

This panel is still under development.

M. Reset

When the left mouse is depressed on the reset option, all changes made on any panels will be reinitialized to the start values. This includes vehicle position and speeds, but does not include any changes made using the five orientation dials above the main menu.

N. Exit

When the left mouse is depressed on the exit option, the simulation is ended and the control is passed back to the operating system of the Iris graphics machine.

O. Inclination

When the left mouse is depressed on this dial, the user is able to rotate the entire viewing angle about the y-axis. On initial start-up of the simulation, inclination is positioned at 17.2 degrees. The box above the dial gives the actual degree position to the nearest 10th of a degree. On the dial, the middle right tickmark is the 0 degree position and the user may adjust the angle from -180.0 degrees to 180.0 degrees.

There are two ways in which the user may use this dial. The first is a snap position where the user depresses the left mouse button anywhere within the dial and the when the button is released, the viewing angle snaps to this degree position. The second way is to depress the left mouse button over the current position of the dial and drag it to the desired angle. The drag method gives a smooth rotation about the axis.

P. Azimuth

When the left mouse is depressed on this dial, the user is able to rotate the entire viewing angle about the x-axis. On initial start-up of the simulation, azimuth is positioned at -51.6 degrees. The box above the dial gives the actual degree position to the nearest 10th

of a degree. On the dial, the bottom tickmark is the 0 degree position and the user may adjust the angle from -180.0 degrees to 180.0 degrees.

There are two ways in which the user may use this dial. The first is a snap position where the user depresses the left mouse button anywhere within the dial and the when the button is released, the viewing angle snaps to this degree position. The second way is to depress the left mouse button over the current position of the dial and drag it to the desired angle. The drag method gives a smooth rotation about the axis.

Q. Distance

1. Near Distance

When the left mouse is depressed on this dial, the user is able to adjust the distance of viewing measured from the left hand corner of the pool. This center dial gives ranges from 7.0 feet to 400.0 feet. This dial enables fine distance adjustments.

The box above the dial gives the actual distance to the nearest 10th of a foot. On the dial, the top tickmark is the zero distance position.

There are two ways in which the user may use this dial. The first is a snap position where the user depresses the left mouse button anywhere within the dial and the when the button is released, the viewing angle snaps to this degree position. The second way is to depress the left mouse button over the current position of the dial and drag it to the desired angle. The drag method gives a smooth rotation about the axis.

2. Far Distance

When the left mouse is depressed on this dial, the user is able to adjust the distance of viewing measured from the lefthand corner of the pool. This dial gives ranges from 0.0 feet to 2000.0 feet. This dial enables rough distance adjustments.

The box above the dial gives the actual distance to the nearest 10th of a foot. On the dial, the top tickmark is the zero distance position.

There are two ways in which the user may use this dial. The first is a snap position where the user depresses the left mouse button anywhere within the dial and the when the button is released, the viewing angle snaps to this degree position. The second way is to depress the left mouse button over the current position of the dial and drag it to the desired angle. The drag method gives a smooth rotation about the axis.

R. Twist

When the left mouse is depressed on this dial, the user is able to rotate the entire viewing angle about the z-axis. On initial start-up of the simulation, azimuth is positioned at 0.0 degrees. The box above the dial gives the actual degree position to the nearest 10th of a degree. On the dial, the top tickmark is the 0 degree position and the user may adjust the angle from -180.0 degrees to 180.0 degrees.

There are two ways in which the user may use this dial. The first is a snap position where the user depresses the left mouse button anywhere within the dial and the when the button is released, the viewing angle snaps to this degree position. The second way is to depress the left mouse button over the current position of the dial and drag it to the desired angle. The drag method gives a smooth rotation about the axis.

LIST OF REFERENCES

[CLOTIER 90]
Clotier, M., Guidance and Control System for an Autonomous Vehicle, Master's Thesis, Naval Postgraduate School, Monterey, CA, June, 1990.

[FLOYD 91a]
Floyd, C. A., et al., "Underwater Obstacle Recognition Using Low-Resolution Sonar," To appear in the Proceedings of the 7th International Symposium on Unmanned Untethered Submersible Technology, September, 1991.

[FLOYD 91b]
Floyd, C.A., Design and Implementation of a Collision Avoidance System for the NPS Autonomous Underwater Vehicle (AUVII) Utilizing Ultrasonic Sensors, Master's Thesis, Naval Postgraduate School, Monterey, CA, September, 1991.

[HARTMAN 89]
Hartman, B.I., et al., "Model and Sensor Based Precise Navigation by an Autonomous Mobile Robot," Proceedings ICAR, 1989.

[HEALY 90]
Healy, A.J., et al., "Mission Planning, Execution, and Data Analysis for the NPS AUVII Autonomous Underwater Vehicle," IARP, First Workshop on Mobile Robots for Subsea Environments, Monterey, CA , 1990.

[JUREWICZ 90]
Jurewicz, T., A Real Time Autonomous Underwater Vehicle Dynamic Simulator, Master's Thesis, Naval Postgraduate School, Monterey, CA, December, 1990.

[KANAYAMA 88]
Kanayama, Y., et al., "A Locomotion Control Method for Autonomous Vehicles," Proceedings of the IEEE International Conference on Robotics and Automation, 1988, pp. 1315-1317.

[KANAYAMA 90]
Kanayama, Y., et al., "A Stable Tracking Control Method for an Autonomous Mobile Robot," Proceedings of the IEEE International Conference on Robotics and Automation, 1990.

[KANAYAMA 91]
Kanayama, Y., Onishi, M., "Locomotion Functions in the Mobile Robot Language, MML," Submitted to the 1991 IEEE International Conference on Robotics and Automation.

[LASALLE 61]
LaSalle, J., Lefschetz, S., Stability by Liapunov's Direct Method, Academic Press Inc., New York, 1961.

[SAVANT 90]
Savant, S., Nonholonomic Trajectory Planning and Nonlinear Feedback Control of an Autonomous Robotic Submersible, Master's Thesis, University of California, Santa Barbara, CA, October, 1990.

[YOERGER 91]
Yoerger, D.R., Slotine, J.E., "Adaptive Sliding Control of an Experimental Underwater Vehicle," Proceedings of IEEE International Conference on Robotics and Automation, 1991.

INITIAL DISTRIBUTION LIST

Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
Dudley Knox Library Code 52 Naval Postgraduate School Monterey, CA 93943	2
Chairman, Code CS Computer Science Department Naval Postgraduate School Monterey, CA 93943	2
Lt. Christopher Magrino 431 Deanview Dr. Cincinnati, OH 45224	2
Dr. Yutaka Kanayama, Code CS/Ka Computer Science Department Naval Postgraduate School Monterey, CA 93943	1
Dr. Yuh-jeng Lee, Code CS/Le Computer Science Department Naval Postgraduate School Monterey, CA 93943	1
Dr. Anthony Healy, Code ME/Hy Mechanical Engineering Department Naval Postgraduate School Monterey, CA 93943	1
Glenn Reid, Code U401 Naval Surface Warfare Center Silver Springs, MD 20901	1

RADM Evans, Code SEA92 1
Naval Sea Systems Command
Washington, DC 20362

Dr. G. Dobeck, Code 4210 1
Naval Coastal Systems Center
Panama City, FL 32407-5000

Dick Blidberg 1
Marine Systems Engineering Lab
SERB Building 242
University of New Hampshire
Durham, NH 03824

Thesis
M27645 Magrino
c.1 Three dimensional
guidance for the NPS
autonomous underwater
vehicle.